AD-A050 965 LOGICON INC SAN DIEGO CALIF F/G 5/2
ADAPT I UNIFORM DATA LANGUAGE (UDL): A FINAL SPECIFICATION.(U)
JAN 78 L E ERICKSON, M E SOLEGLAD N00014-76-C-0899
UNCLASSIFIED 76-C-0899-1 NL

1 OF 2
AD
A050 965

Report 76-C-0899-1

ADAPT I UNIFORM DATA LANGUAGE (UDL): A FINAL SPECIFICATION

Logicon, Inc.
4010 Sorrento Valley Blvd. P.O. Box 80158
San Diego, California 92138

30 January 1978

Final Report for Period 1 November 1977 – 30 January 1978

DOD DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 Wilson Boulevard
Arlington, VA 22209

OFFICE OF NAVAL RESEARCH
INFORMATION SYSTEMS PROGRAM
Code 437
Arlington, VA 22217

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 76-C-0899-1	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ADAPT I Uniform Data Language (UDL): A Final Specification	5. TYPE OF REPORT & PERIOD COVERED: Final Report, 1 Nov 77 – 30 Jan 78 6. PERFORMING ORG. REPORT NUMBER: 76-C-0899-1
7. AUTHOR(s): L. E. Erickson, M. E. Soleglad, S. L. Westermark	8. CONTRACT OR GRANT NUMBER(s): N000014-76-C-0899
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Logicon, Inc., 4010 Sorrento Valley Boulevard, P.O. Box 80158, San Diego, California 92138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209	12. REPORT DATE: 30 Jan 78 13. NUMBER OF PAGES: 150 p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research Information Systems Program, Code 437, Arlington, Virginia 22217	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Base Management Systems (DBMS)	Intelligent Terminal	UNIX
Very Large Data Bases (VLDBs)	ARPANET	
Query Languages	Network	
Uniform Data Language (UDL)	Language Transformations	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Under the ADAPT project, a prototype intelligent terminal will be developed which provides users and/or other systems a uniform interface for accessing multiple on-line data bases located on different systems. The underlying technology applied by ADAPT will be the transformation from one uniform data language, UDL, to other target query languages which reside on a network. ADAPT will be comprised of four phases: ADAPT I provides the fundamental ADAPT system architecture baseline for subsequent phases and also a limited user data base query and display facility →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 55 IS OBSOLETE

406012

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT

for four data base management systems. ADAPT II will provide users a data base maintenance facility for four DBMSs, as well as data display enhancements. ADAPT III will produce two production models of ADAPT, one installed in network centers and the other as a stand-alone intelligent terminal. ADAPT IV will provide a local data base manager for onsite file creation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

INTRODUCTION

The requirement for data sharing occurs in many computer application areas. Examples are hospital networks, banking systems, reservation systems, and government information systems. These systems accumulate large volumes of information which are of interest to a large community of users. As a result, the large data bases become much too large for feasible duplication for various applications. In addition, utilizing the same information for different applications often necessitates restructuring of data, a process that is both time consuming and expensive. Therefore, there is a growing need for online and real-time data sharing. This requirement is best satisfied by the facilities provided with computer networking. As a result of computer networking, Very Large Data Bases (VLDBs) are logically feasible where each local data depository (at a given network node) can be thought of as a small logical data set of the VLDB. It seems reasonable to speculate here that most VLDBs to be constructed over the next five years will be formed through computer networking of smaller resident data base management systems. This seems inevitable due to the excessive costs associated with alternatives to networking; i. e., data base duplication and/or restructuring.

Although computer networking is a viable solution to the growing need of online data sharing, it also creates a problem with respect to the actual utilization of dissimilar data bases. This problem can best be described as a "multi-user interface" problem. The user interfaces referred to here are those of data standards, sign-on procedures, data base query languages, and logical data structures. Data standards

refer to the form in which data are represented; i. e., how they are placed into the system and in what form they are displayed. User sign-on procedures refer to the "login" sequences that each data base management system requires; i. e., how one acquires access to a data base management system as well as to the individual data bases contained in it. Logical data structures are those structures in which a user views the data, as opposed to a physical data structure; i. e., the form in which the data are physically stored in the system. Finally, data base query languages are the sets of statements, commands, and control sequences which a user employs to access a data base. Almost always, a query language is couched around a set of logical data structures; i. e., a user's queries are constructed in an environment composed of logical data structures.

An inherent problem with data bases shared via computer networking is the proliferation of these user interfaces. Each interface must be mastered by individual users of the network. This problem is well recognized and has been the subject of a number of studies. Recently, Logicon completed a study of a multi-language problem occurring in the Community Online Intelligence Network System (COINS). Currently, a COINS user must employ three different user data languages to query the 48 intelligence data files provided through COINS I. COINS II, which utilizes an ARPANET technology, is scheduled to be expanded by one new system per year so that the intelligence community will be able to access 100 – 150 files supported by 8 – 10 systems within the next five years. Unfortunately, each of these systems has its own query language, with the result that COINS users will be required to learn 8 – 10 query languages if they wish to query these files. Further, because each data base query language is implemented using a different data base management system, even the interpretation

for a query to duplicate files on different systems will vary. Finally, because each system supports different types of terminals, the user is forced to learn terminal-unique characteristics in order to access different systems within the community.

ADAPT, which stands for ARPA Data Base Access and Presentation Terminal, forms a basis for a production prototype intelligent terminal which will provide network users and/or other systems a uniform interface for accessing online data bases on a multitude of dissimilar systems. Initially the ADAPT system (which is briefly described in the next section) has been developed with a particular network in mind (i. e., COINS); however, it must be stressed here that the technologies developed and refined during this effort are portable to other similar "multi-user interface" problems that currently exist on other networks; i. e., ARPANET, Intelligence Data Handling Systems (IDHS), etc.

The remainder of this report consists of five additional sections and an appendix. The next section describes the ADAPT project and the four phases proposed for it during its on-going development. The third section provides a specification of the ADAPT I Uniform Data Language (UDL), including its logical data structures and statement and command set. This section presents the data base query language available to ADAPT I users. The fourth section provides a specification of the ADAPT I Data Definition Language (DDL) that is available to the ADAPT I superuser for data base definition. The fifth section provides a specification of the ADAPT I Transformation Definition Language (TDL) available to the ADAPT I superuser for transformation specific data definition. The last section details the error diagnostics and messages that might occur during the operation of ADAPT I.

Finally, an appendix (appendix A) is provided which presents a specification for an expanded UDL statement set that is envisioned for the completed ADAPT system, phases I – IV.

Although the primary purpose of this document is to present a specification of the Uniform Data, Data Definition, and Transformation Definition Languages supported by ADAPT I, it seems appropriate to provide adequate introductory material for those readers who are not familiar with the proposed ADAPT system and its inception.

ADAPT DESCRIPTION

Originally the ADAPT project was conceived as being comprised of three developmental phases, ADAPT I, ADAPT II, and ADAPT III. The first two phases were to be production prototypes only, the actual production model not appearing until the third phase of development. The purpose of this section is to briefly define each ADAPT phase, outlining the hardware/software suite to be used in its development as well as presenting the user's capabilities provided with each. A new phase, ADAPT IV, is also proposed.

The ADAPT system is being developed in San Diego, at the Tactical and Training Systems Division of Logicon. The hardware configuration currently utilized for this phased development consists of the following devices:

a. Digital Equipment Corporation (DEC) PDP-11/45 with 65K parity core memory.

b. LA36 DECWRITER II console.

c. RK11 1.2M-word RK05 disk drive and controller.

d. Two RK05 1.2M-word disk drives.

e. Tally 4300, 300-lpm (96-character ASCII) medium speed printer.

f. Three MiniBEE-4 CRT terminals.

g. ACC Very-Distant-Host Interface (VDH-11C) for ARPANET connection.

Although this equipment suite is somewhat modest, it does provide an excellent environment for initial ADAPT development.

ADAPT I has been developed as processes under UNIX (an operating system developed at Bell Laboratories). UNIX and its accompanying subsystems furnish ADAPT developers an excellent set of software including a sophisticated time-sharing environment, compiler-compilers, various compilers and assemblers, sophisticated text editors, debugging packages, etc. The major processes developed for ADAPT I have been implemented using C language. Also, the LR(1) compiler-compiler (YACC) has provided much of the front-end processing for ADAPT. The ADAPT development center in San Diego is a "very-distant-host" on the ARPANET.

ADAPT I's purpose is essentially three-fold: first, the development of the fundamental ADAPT system architecture within a PDP-11/UNIX environment; second, the development of a basic language transformation technology; and, third, the demonstration that language transformations as developed can be exercised properly through network protocols.

Although the ADAPT I effort is a very important and extensive phase, much of the software "products" produced during this phase are invisible to the ADAPT I user. The actual user interfaces that are supported in ADAPT I are relatively minimal, with most software technology being applied to developing the internal fundamental system architecture that forms the basis for subsequent phases. However, from a user's point of view, ADAPT I provides a limited but useful set of capabilities. First of all, as was stated earlier in the introduction, the actual ADAPT I implementation is based on a particular network and on a fixed set of data base management systems resident on this network. The network is the Community Online Intelligence Network System (COINS), and the four data base management

systems residing on it are the DMS-1100/QLP system, the Defense Intelligence Agency On-Line System (DIAOLS), the SIGINT On-Line Information System (SOLIS), and the Technical Information Processing Systems (TIPS/TILE). Based on the results from a previous contract, Logicon has developed a set of language transformations which map a single data base query language (UDL, a language developed by Logicon) to a set of eight different data base query languages. At the time of this analysis, three of the eight languages were installed on the COINS network and the remaining five were future candidates for network residency. This set of language transformations involved three separate areas: logical data structure transformations, data base query transformations, and data base maintenance transformations. ADAPT I provides implementations of the transformations for the first two areas: data structures and queries. The third set of transformations will be provided in ADAPT II. The next section of this report presents a specification of the user language, UDL, that is required to support the four initial transformations provided in ADAPT I. As can be readily seen, ADAPT I UDL is somewhat complete as far as data structures and data base query facilities are concerned, but is essentially void in areas dealing with report generation, data base maintenance and data manipulation.

Another important user interface supported in ADAPT I is a facility for redefining existing target system files into logical structures compatible with UDL. This facility is called the Data Definition Language (DDL) and is detailed in the fourth section of this report.

Required with the schemas generated by DDL are a set of transformation files which provide host-specific file information required for the transformations. The Transformation Definition Language (TDL) subsystem provides this interface and is detailed in the fifth section of this document.

Finally, ADAPT I provides limited but useful facilities for displaying data received from a given target system (as the consequence of a previous UDL query), the ability to display the logical structure of a file, and other more system control-oriented facilities.

ADAPT II is a significant upgrade to ADAPT I as far as direct user facilities are concerned, although it is still considered only as a production prototype. One of the major products of the ADAPT II effort will be the completion of the language transformation set; i. e., the inclusion of a set of language transformations for data base maintenance facilities. These transformations, as in ADAPT I, will be for the four target systems listed previously. Other ADAPT II features are:

a. Sophisticated display and report generation facility.

b. Local record sort facility.

c. Expanded data structure mapping facility.

d. Complete file guide capability.

e. Improved user access control interfaces.

As the new ADAPT II products are developed, the UDL developed in ADAPT I expands accordingly to accommodate the new user interfaces. Appendix A presents a complete specification of UDL, a UDL that will become fully realized at the completion of ADAPT III. As can be readily seen, the ADAPT I UDL is a direct subset of the language specified in appendix A.

ADAPT III will provide yet additional user interfaces, although its main purpose is to convert the production prototype system as developed in ADAPT II to two production models. The first production model will be installed at various network service centers, and the

second model will exist as a free-standing intelligent terminal implemented using then-current microprocessor and "small" mass memory technology. The user interface improvements developed in ADAPT III will provide a complete data manipulation capability for the user. This will include programming language-like facilities, such as local variable declarations, numeric expression and assignment statement facilities, and program and loop control facilities. Refer to appendix A, paragraph A. 4. 5, for more information.

ADAPT IV, as conceived here, will provide ADAPT users with a local data base management capability. It must be remembered that, although the first three phases of ADAPT supply the user with a complete uniform data base management query language, users can not define and create new files at their installations. All files physically reside at the various target systems throughout the network. It is interesting to point out here that the same Data Definition Language (DDL) provided in ADAPT I can be used to define new files which will exist locally at the user's installation. Of course, what will be required for ADAPT IV is a data base manager to manipulate these files (i. e., local data base interrogations, display, maintenance, backup/recovery, etc.) and mass storage to accommodate the local files.

The remaining sections constitute a detailed specification of the ADAPT I UDL statement and command set and the DDL and TDL subsystems. Although this document is a language specification and not a user's manual per se, it is complete such that an individual familiar with computers and one or more programming languages can utilize it as the latter.

UNIFORM DATA LANGUAGE (UDL) FOR ADAPT I

Introduction

This section provides a specification for the subset of the Uniform Data Language (UDL) available for ADAPT I. The commands available for ADAPT I are also described in this section, even though they are not considered a true part of the UDL proper. Although commands are an important integral of any system, they are usually implementation-dependent, since their utility is very specialized. In line with this, one can usually think of UDL data structures and statements in more abstract terms, essentially removed from implementation considerations.

The format of this section essentially parallels that of appendix A, but of course is considerably briefer since the initial ADAPT I UDL is a small subset of the complete UDL.

Briefly, one can see that a fairly complete data structure complement is being provided in ADAPT I as well as a complete data base interrogation capability. However, the remaining capabilities are limited to a somewhat restricted display facility with no data base maintenance or data manipulation interfaces provided. These missing capabilities, as mentioned earlier in this report, will be provided in subsequent ADAPT phases.

The information which follows is divided into four basic groups:

a. Basic definitions.

b. Data structures.

c. UDL statements.

d. ADAPT I commands.

Basic definitions provides descriptions of the character set and the construction of names and labels. Data structures are described with respect to their logical structure, designated data types, and assigned data attributes. UDL statements and ADAPT I commands are described and their syntax and semantics provided in detail.

Basic Definitions

This paragraph presents the fundamental definitions for the ADAPT I UDL. Included are character set, data name, and label definitions, and descriptions of the three data constants. These definitions, and their syntax and semantics, will be referenced throughout the remainder of this section.

CHARACTER SET – The American Standard Code for Information Interchange (ASCII) is the character set adopted for ADAPT I UDL. Out of this character set, the following delimiters, digits, and letters are valid for ADAPT I UDL statement construction.

Syntax –

<delimiter> = + | - | / | (|) | , | . | ' | Δ | % | ; | CR

<digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Semantics – The meaning of each of the delimiters is as follows:

+: unary plus operator.

-: unary minus operator.

/: geographic constant delimiter.

(: list or grammatical enclosure (initial).

): list or grammatical enclosure (terminal).

,: list separator.

.: decimal point.

': delimit (initial and terminal) character constant.

Δ: designates a space.

%: denotes geographic constant.

;: terminates statements and commands

CR: carriage return (newline)

Decimal digits are used to express numeric constants, to form labels and names (except as the first character), and to form character and geographic constants.

A letter may be any letter of the alphabet. Letters may be used in labels, names, and in character and geographic constants.

NAMES AND LABELS – Names and labels are ADAPT I UDL identifiers which are constructed by the user. Following are the syntax and semantics of UDL names and labels.

Syntax –

<name> = <name> <letter> | <name> <digit> | <letter>

<label> = <name>

<tag> = <label>

Semantics – A name can consist of up to eight characters, where these characters must be either letters or digits. The first character must be a letter. Names are used to identify various entities in an ADAPT I UDL statement complex. All syntactically correct names are valid constructs in ADAPT I UDL except for those occurring in the

reserved word list in table 1. For obvious reasons, the complete reserved word list for all of UDL is shown here (Note: entries without an asterisk are applicable for ADAPT I).

Names can be utilized in two basic ways: pure names or as part of other identifiers. Pure names can occur as either a list name or as a data base structure name; i. e., file-name, field-name, etc. Names can also be used to form labels.

Labels are user-defined identifiers that are located in the front of ADAPT I UDL statements. The use of labels is optional except where otherwise noted. Labels that are referenced in ADAPT I UDL statements are called tags.

TABLE 1. ADAPT I UDL RESERVED WORD LIST.

ACTION	DECLARE*	FIND	LT	POSITION	TAB*
ADD*	DELETE	FORMAT*	MAX*	PROCEDURE*	TABSET*
ALL*	DEP	GE	MODE	PULL*	THEN*
ALONG	DESCEND*	GEO	MV	PUT*	TO*
AND	DISPLAY	GOTO*	NKEY	QUIT	TRAILER*
ARRAY	DO*	GT	NOR	REC*	TRANFILE
AT*	E*	HEADER*	NOT	RECORD*	TRANS
ATT	EARRAY	HSP*	NUM	REMOTE	TREE
BATCH	ECHO	IF*	NUMERIC*	REMOVE*	TRESET*
BY*	ELSE*	IN	OCC*	RETURN*	USE*
CALL*	END*	INPUT*	OCCURRENCE*	RGROUP	V
CAT*	EPROC*	INSIDE	OFF*	ROUTE	VAL*
CHANGE*	EQ	INTER	ON*	RTYPE*	VALIDATE
CHAR	ERGROUP	INV	OPEN	SAVE	VALUE*
CIRCLE	ESCHEMA	KEY	OR	SCHEMA	VERT*
CLEAR*	ETRAN	LABEL	ORG	SKIP*	VIEW
CLOSE	EXECUTE	LE	OUTPUT*	SORT*	VIS
CONTAINS*	EXIST*	LINE*	OUTSIDE	SOURCE	WORD*
CREATE*	EXIT*	LIST	PAGE*	SPACE*	WRG
CURRENT*	FIELD	LOCAL	POLY	SV	XOR
DATABASE	FILE	LOCSC*	POS		

DATA CONSTANTS AND EXPRESSIONS – This paragraph specifies the syntax and semantics for all legal ADAPT I UDL data constants and expressions. Three basic data constants are recognized: character-constants, numeric-constants, and geographic-constants.

Syntax –

<data-constant> = <character-constant> | <numeric-constant> | <geographic-constant>

<character-constant> = '<characters>'

<characters> = <letter> <characters> | <digit> <characters> | <delimiter> <characters> | <letter> | <digit> | <delimiter>

<numeric-constant> = <numeric-clause> | <sign> <numeric-clause>

<numeric-clause> = <integer> · <integer> | <integer> · | · <integer> | <integer>

<integer> = <digit> <integer> | <digit>

<sign> = + | -

<geographic-constant> = % <lat-term>Δ/<long-term>

<lat-term> = <position-term1> <direction1>

<position-term1> = <degree1> Δ <min-sec> | <degree1>

<degree1> = <digit> <digit>

<min-sec> = <min> Δ <sec> | <min>

<min> = <degree1>

<sec> = <degree1>

<direction1> = N | S

<long-term> = <position-term2> <direction2>

<position-term2> = <degree2> Δ <min-sec> | <degree2>

<degree2> = <digit> <degree1>

<direction2> = E | W

<expression>	=	<field-term> | <file-name> | <array-name> | <repeating-group-name> | <list-name> | <data-constant>
<field-term>	=	<field-name> (<subscript-list>) | <field-name>
<field-name>	=	<name>
<subscript-list>	=	<subscript>, <subscript>, <subscript> | <subscript>, <subscript> | <subscript>
<subscript>	=	<integer>
<file-name>	=	<name>
<array-name>	=	<name>
<repeating-group-name>	=	<name>
<list-name>	=	<name>

Semantics – For ADAPT I UDL data structures, character-constants are applicable for character data types only. Numeric-constants are valid in data structures with numeric data types only, and geographic-constants are valid in data structures with geographic data types.

Expressions in ADAPT I UDL are composed only of names and data-constants. In later ADAPT phases, full algebraic expressions will be made available.

Data Design

This paragraph describes the data structures, data types, and data attributes recognized in ADAPT I UDL. There exist five primary data structures in ADAPT I UDL:

a. Field.

b. Aggregate.

c. Record.

d. File.

e. Data base.

Data types are associated with the lowest level data structure, the field, and three basic types are defined:

a. Character.

b. Numeric.

c. Geographic.

Along with data types, access attributes are also associated with each field, and control access with respect to interrogation and display.

The remainder of this paragraph discusses the foregoing in detail.

DATA STRUCTURES – Five primary data structures are recognized in ADAPT I UDL: data bases, files, records, aggregates, and fields. The most important structures with respect to user utilization and overall ADAPT I UDL design considerations are fields and aggregates. These two structures will be discussed first.

Fields – Two field structures are recognized in ADAPT I UDL:

a. Single-valued field.

b. Multivalued field.

The two fields are discussed in the following paragraphs, where their logical structure and rules governing their access with respect to interrogation and display are described.

Single-Valued Field – A single-valued field has the logical structure depicted in figure 1. It has a name with which is associated a

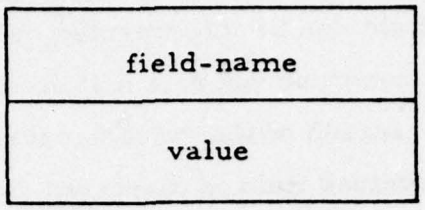


Figure 1. Logical Structure of a Single-Valued Field.

single value occurrence. All references to a single-valued field must reference its name. The value associated with a single-valued field must conform to the data-type defined for that field.

A single-valued field can be interrogated by referencing its name against some relational condition where a match is successful depending on its single-valued occurrence.

Multivalued Field – A multivalued field has the logical structure depicted in figure 2. It has a name with which is associated a set of value occurrences. All references to a multivalued field must reference its name. Each value occurrence of a multivalued field must conform to the data-type defined for that field. The number of value occurrences that can be contained in a multivalued field is arbitrary, and either no occurrences or many occurrences are possible.

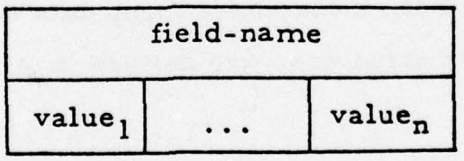


Figure 2. Logical Structure of a Multivalued Field.

A multivalued field can be interrogated by referencing its name against a relational condition where a match is successful depending on a value-by-value search of the value occurrence set contained in the field. When a multivalued field is displayed, all of its occurrences are shown.

Aggregates – An aggregate is a named collection of fields and/or other aggregates. This named collection of data structures can occur an arbitrary number of times within a record. One occurrence of such a collection is called a data set. Since aggregates may contain other aggregates, complex hierarchical structures can be formed with them. A collection of data sets belonging to the same aggregate and having the same ancestral occurrence of a parent aggregate is called an aggregate instance.

In most cases, an aggregate is a structural convenience since the aggregate itself cannot be referenced specifically for reading or writing data. There exist exceptions to this, however, where aggregates can be referenced as a totality. Generally these exceptions deal with convenience modes in data display.

Aggregates do not have specific data types or data attributes associated with them. Since an aggregate can have an arbitrary number of fields defined for it, it may have many data types and data attributes. Two aggregate data structures are defined in ADAPT I UDL:

a. Array.

b. Repeating group.

The following paragraphs discuss these aggregates in detail.

Array – An array is a named multivalued data structure which can be composed of any number of fields or arrays. An array has the logical structure depicted in figure 3. Repeating groups cannot be part of an array definition. At least one field must be defined in an array. An arbitrary number of occurrences of an array data set is legal, but the maximum number of occurrences must be established as part of the array's definition.

An array cannot be interrogated as a totality, since references are restricted to actual fields defined for that array. For each array defined in a record, an index is associated with it. This index has the range 1 – n where n is the number of occurrences defined for that array. All query references to fields defined in an array must be accompanied by this index. Since arrays can be nested, other implied subscripts are also required. That is, if a given array is defined under two

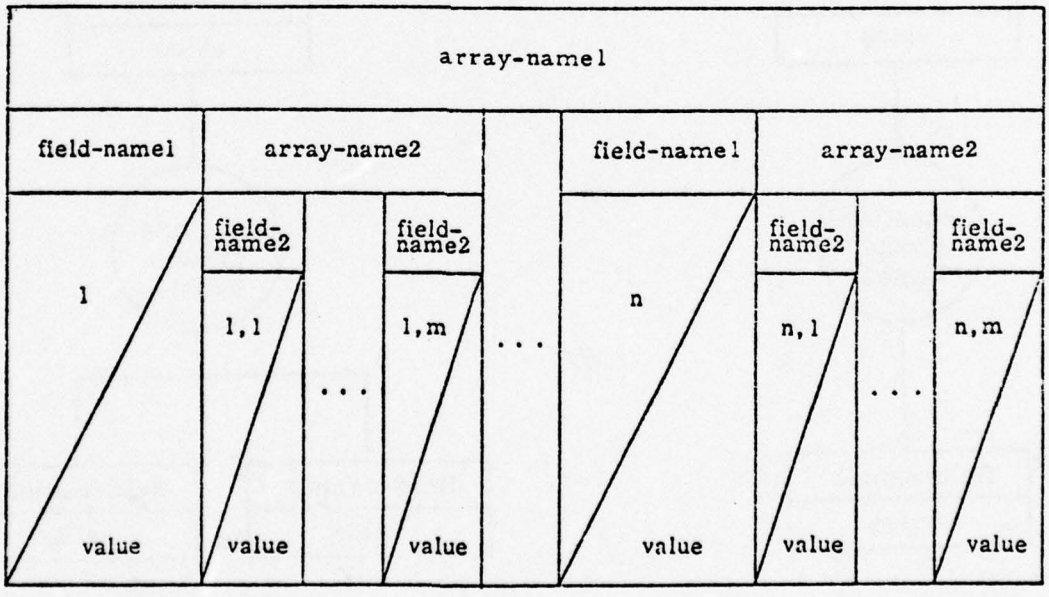


Figure 3. Logical Structure of an Array (two array levels are shown).

other arrays, three subscripts must be specified in its field references. Array-name references are only legal for display where an entire array can be displayed.

Repeating Group – A repeating group is a named multivalued data structure which can be composed of any number of fields, arrays, or other repeating groups. A repeating group's logical structure is depicted in figure 4. At least one field must be defined in a repeating group. An arbitrary number of occurrences of a repeating group data set is legal.

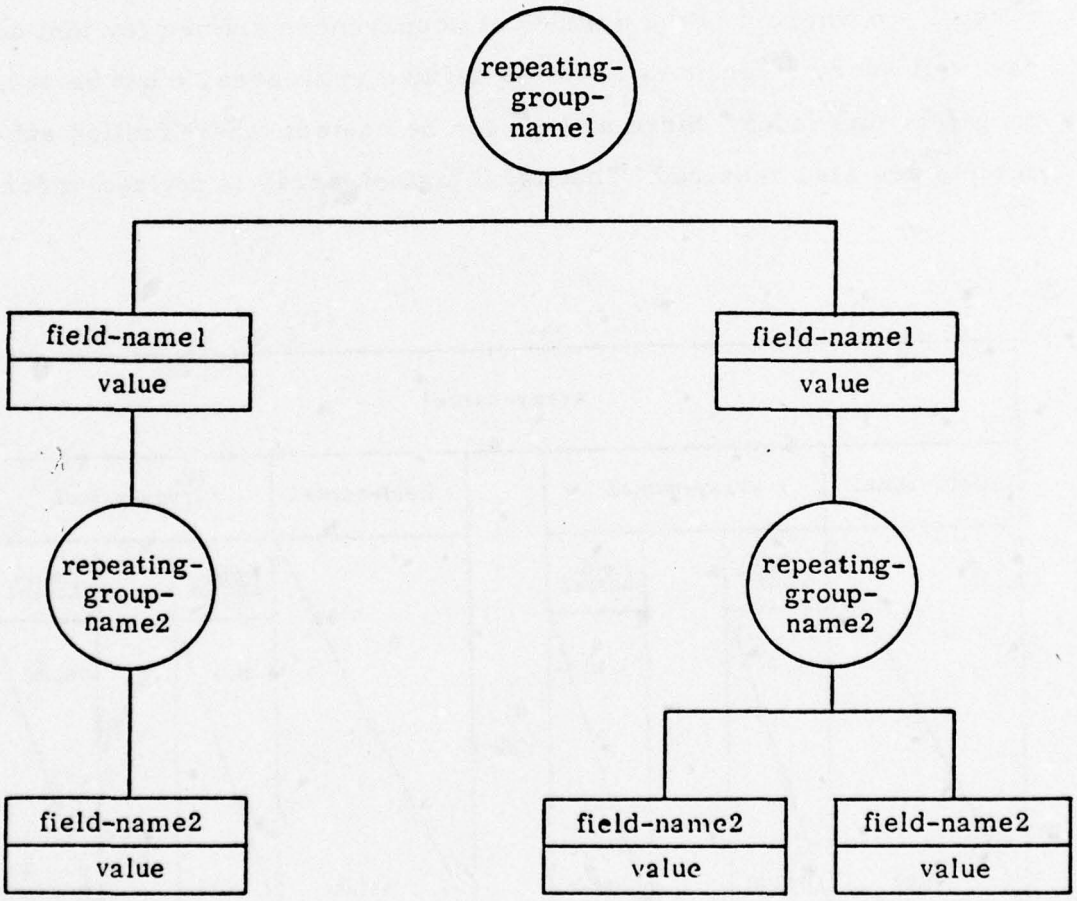


Figure 4. Logical Structure of a Repeating Group (two repeating group levels are shown).

A repeating group cannot be interrogated as a totality, since references are restricted to actual fields defined for that repeating group. Since repeating groups are pure multioccuring data structures, an arbitrary number of occurrences is allowed, and this number may be variable for each record (the size of an array is fixed for all records).

Record – Records are probably ADAPT I UDL's next most important data structure since they form the physical unit of selectability. Individual records are not named. Records are defined by the fields and aggregates defined under them. All records in a file have the same logical structure containing the same fields and aggregates. This does not necessarily imply that all records are the same size; true multioccurring data structures such as multivalued fields or repeating groups can have an arbitrary number of occurrences for any record.

Files – A file is a named collection of records. A file is the largest data structure which an interrogation statement can reference.

Data Base – A data base is a collection of files, and can be roughly thought of as synonymous with a given target system.

DATA TYPES – Data types designate the nature of the data that a field contains. ADAPT I UDL data types can be of three basic types: character, numeric, and geographic. Fields containing character data can be referenced with the EQ relational operator only. Numeric data types specify that a field contains data of a purely numeric nature. For fields designated with a numeric data type, the EQ operator plus all the standard relational operators (i.e., GT, LT, etc.) are valid. The special geographic data type is for fields that contain geographic data. Geographic data always contain the positional dyad

latitude/longitude. The special geographic operators, INSIDE, OUTSIDE, and ALONG, are used exclusively with fields designated with a geographic data type.

All ADAPT I UDL fields must have a designated data type. All constants referenced in conjunction with these fields must conform to this data type. Refer to the definition of ADAPT I UDL data constants.

DATA ATTRIBUTES – Data attributes, as data types, are associations placed on fields. Data attributes affect a field's operability with the ADAPT I UDL statement set. Two attribute classifications must be designated for an ADAPT I UDL field:

a. Interrogation.

b. Display.

The interrogation attribute can be one of three kinds: keyed, nonkeyed, and dependent. Fields with a data attribute of keyed can be referenced in any FIND statement. That is, there are no restrictions to their use in selection criteria. A field designated as dependent can also be utilized in a FIND statement. However, this must be a dependent FIND statement where its record source is from a previous FIND statement. Therefore, a field designated as dependent cannot be referenced in an initial FIND statement. Fields designated as nonkeyed cannot be referenced in a FIND statement.

The display attribute can be one of two kinds: visible or invisible. Fields designated with a data attribute of visible can be displayed via the DISPLAY statement. Fields designated invisible cannot be displayed at all. A field cannot be simultaneously invisible and nonkeyed since the field could be neither queried nor displayed, i. e., could not be referenced in any way.

ADAPT I UDL Statements

INTRODUCTION – For the first phase of ADAPT, ADAPT I, only two UDL statements are available: the FIND statement and the DISPLAY statement. The FIND statement provides ADAPT I users with a fairly complete data base interrogation facility (as is apparent when compared to paragraph A. 4. 2 in appendix A).

On the other hand, the DISPLAY statement provided for ADAPT I gives users only a limited display capability, restricted to UDL default format only. Although this display facility is somewhat restricted by lacking user-specified format control, it does provide the user with a means of displaying desired data in an orderly fashion.

INTERROGATION STATEMENT – Interrogation statements in ADAPT I UDL are used to isolate records from a file by specifying a set of selection criteria against that file. Interrogation does not display a file but only isolates records for subsequent display. This paragraph describes the syntax and semantics of ADAPT I UDL's interrogation facility, the FIND statement.

Syntax –

<find-statement> = FIND <source-clause> <selection-criteria>;

<source-clause> = IN <file-name> | SOURCE (<tag>)

<selection-criteria> = <selection-clause> OR <selection-criteria> | <selection-clause>

<selection-clause> = <selection-phrase> AND <selection-clause> | <selection-phrase>

<selection-phrase> = <selection-factor> <log-op> <selection-phrase> | <selection-factor>

<log-op> = XOR | NOR

<selection-factor> = NOT <selection-factor> | <selection-primary>
<selection-primary> = <scope-qualifier> (<selection-criteria>) |
(<selection-criteria>) | <selection-term>
<scope-qualifier> = <repeating-group-name>
<selection-term> = <rel-term1> | <rel-term2> | <rel-term3>
<rel-term1> = <field-term> <rel-op1> <data-constant>
<rel-op1> = EQ | GT | LT | GE | LE
<rel-term2> = <field-term> <rel-op2> <numeric constant>,
<numeric constant>
<rel-op2> = WRG | ORG
<rel-term3> = <field-term> < rel-op3> <geo-exp>
<rel-op3> = INSIDE | OUTSIDE | ALONG
<geo-exp> = <circle-exp> | <poly-exp> | <route-exp>
<circle-exp> = CIRCLE (<radius-term>, <geographic
constant>)
<poly-exp> = POLY (<lat-long-list>)
<route-exp> = ROUTE (<band-term>, <lat-long-list>)
<lat-long-list> = <geographic-constant>, <lat-long-list> |
<geographic-constant>
<radius-term> = <numeric-constant>
<band-term> = <numeric-constant>

Semantics – The source-clause must specify either a file-name or a statement label of a previous FIND statement. FIND statements which reference other FIND statements are called dependent FIND statements since they isolate records from a set of records previously isolated by another FIND statement. The selection-criteria is a specialized boolean-expression used exclusively

for record selection. Legal boolean operators for selection-criteria are OR, AND, XOR, NOR, and NOT, and their definition and operation precedence is as follows:

a. OR – boolean "inclusive-or" condition (union).

A	OR	B
0	0	0
0	1	1
1	1	0
1	1	1

b. AND – boolean "and" condition (intersection).

A	AND	B
0	0	0
0	0	1
1	0	0
1	1	1

c. XOR – boolean "exclusive-or" condition.

A	XOR	B
0	0	0
0	1	1
1	1	0
1	0	1

d. NOR – boolean "joint-denial" condition (not-or).

A	NOR	B
0	1	0
0	0	1
1	0	0
1	0	1

e. NOT – boolean "complement" condition (negation).

NOT	A
1	0
0	1

Boolean operator precedence is the following:

a. NOT.

b. XOR, NOR.

c. AND.

d. OR.

The boolean operators XOR and NOR are evaluated "left-to-right." The use of parentheses is valid in selection-criteria and forces precedence inside their scope. Parentheses may be nested and, if so, evaluation proceeds from the innermost parenthesis level outwards.

The argument of a boolean term (called a selection-primary) can be either a selection-term, which is composed of three relational term types, or other selection-criteria enclosed in parentheses with an optional scope-qualifier. For any given set of selection-criteria, the maximum scope of operation is over a record. This is called a selection-criteria's implicit scope. For complex hierarchical structures within a record, it is also desirable to be able to limit scopes of operation to various subtrees of the structure. The scope-qualifier has been designed to do this.

The scope-qualifier must be the name of a repeating group. A selection-criteria qualified by a scope-qualifier is forced to satisfy that criteria for the same occurrence of the named repeating group. This also implies that the field-terms referenced in the qualified

selection-criteria must be defined directly for the named repeating group. The scope-qualifier will affect the selection process result only if there exist two or more selection-primaries as arguments of the scope-qualifier such that one or more of the logical connectors is an AND (after the selection-criteria has been reduced to disjunctive-normal-form).

Scope-qualifiers can also be nested to force the desired lineage path during the selection process.

Since the scope-qualifier is only applicable for criteria which are connected by an AND operator, complex selection-criteria can be difficult to analyze with respect to a scope-qualifier. Complex selection-criteria under the scope of a repeating group can be reduced to a more understandable form with the application of three rules:

a. Reduce the selection-criteria to disjunctive-normal-form.

b. Distribute the scope-qualifier through the various disjuncts.

c. Drop the scope-qualifier from any single-term disjunct.

The result is a series of scope-qualified terms or unqualified single-terms, connected by OR-operators. (note: steps b-c above convert the expression to a scope-normal-form.

Relational terms can be divided into three basic groups:

a. Standard relational terms.

b. Range relational terms.

c. Geographic relational terms

Standard Relational Terms – Five standard relational operators are legal in ADAPT I UDL selection-criteria (see figure 5):

LT – data-specification less than numeric-constant.

GT – data-specification greater than numeric-constant.

LE – data-specification less than or equal to numeric-constant.

GE – data-specification greater than or equal to numeric-constant.

EQ – data-specification equals data-constant (numeric or character).

Range Relational Terms – Two operators are provided for range operations: WRG and ORG. The WRG operator requires that the field-term be within or equal to the limits established by the two specified numeric-constants. The first expression specifies the lower range limit, and the second expression specifies the upper range limit. The ORG range requires that the field-term be outside the range established by the two numeric-constants.

Operator	Data Types		
	Character	Numeric	Geographic
EQ	Yes	Yes	–
LT, GT, LE, GE, WRG, ORG	–	Yes	–
INSIDE, OUTSIDE, ALONG	–	–	Yes

Figure 5. Legal Combinations of Data Types and Selection-Criteria Relational Operators.

Geographic Relational Terms – Geographic relational terms apply to field-terms (with a geographic data type) and their relationship to a specified set of circles, routes, or n-sided polygons on the surface of the earth. Three geographic operators and three specialized geographic-expressions are recognized in UDL.

The INSIDE operator requires that a field-term's geographic value (latitude/longitude constant) be inside the circle or the polygon specified by the geographic-expression. Similarly, the OUTSIDE operator requires that a field-term's value be outside the polygon specified, and the ALONG operator requires that the value specified in the field-term be on the perimeter of the specified route. The specialized geographic-expression can appropriately specify either a circle, an n-sided polygon, or a multiple directioned route. For a circle, both the center and radius must be established. A route is established by specifying the points it lies on and the width of the route. Polygons are determined by a list of latitude/longitude specifications. Refer to the data-constant description for the syntax and semantics of geographic-constants.

DISPLAY STATEMENT – The data presentation facility provided in ADAPT I is limited to UDL default format. In this version, the user will neither be able to establish headers or trailers nor generate sophisticated reports tailored with user-specified format declarations. Following are the syntax and semantics for the ADAPT I UDL DISPLAY statement.

Syntax –

<display-statement> = DISPLAY <display-clause>;

<display-clause> = SOURCE (<source-arg>) <display-list> |
SOURCE (<source-arg>)

<display-list> = <display-element>, <display-list> | <display-element>

<display-element> = <field-term> | <aggregate-name> | TREE (<aggregate-name>) | <character-constant>

<aggregate-name> = <repeating-group-name> | <array-name>

<source-arg> = <tag> | <listname>

Semantics – The DISPLAY statement operates on a list of records. These records may be on a user-specified list or they may be on an implicit list generated by a FIND statement. If a display-list is not specified, the entire record(s) is displayed.

The display-list is a list of data structures below the record level which provides the user with a selective and ordered mechanism for displaying portions of a record. The fundamental element of a display list is the display-element. Display-elements can be one of four basic kinds: a character-constant, a field-term, an array-name, or a repeating-group-name. In addition, an entire repeating group or array may be displayed by using the TREE qualifier. If a display-list is present, at least one display-element other than character-constant must be specified.

Each display-element is displayed in UDL default display format. This format provides a simple means for a user to display fields and aggregates of records. Figure 6 illustrates the actual UDL default format as it applies to the various kinds of fields and aggregates. Due to the inherent nature of hierarchical structures, records which contain levels of aggregates will be displayed with the tree structure intact, regardless of the order of the specified display-items. Data structures defined for the same aggregate, however, will be displayed in the order they are specified in the display-list. An aggregate display-element will cause the display of all occurrences of the specified aggregate at its data set level only; that is, subordinate aggregates will not be displayed. An aggregate display-element qualified by TREE behaves as the situation just described except that all subordinate aggregates are also displayed (as shown in figure 6).

If a list of field-terms is specified as a set of display-elements and they are embedded in a hierarchical structure of aggregates, the entire upper tree structure which encompasses all the specified field-terms will be displayed. Aggregate names will also be displayed as node markers, and will be inserted into the proper places of the tree. Combinations of field-terms, repeating-group-names, and array-names may be specified in the same display-list. The display for each display-element behaves as shown in figure 6. Each record displayed will contain the minimum tree structure which encompasses the display-elements. As brought out earlier in this section, if a display-list is not specified, the entire record is displayed. Thus if aggregates are present in the record definition, the entire tree structure will be displayed.

```
Single-Valued Field
    field-name = value
Multivalued Field
    field-name = value1
               = value2
               :
               = valuen
Array (TREE (array-name1))
    array-name1 (1)
        field-name = value
        array-name2 (1, 1)
            field-name = value
            field-name = value
        array-name2 (1, 2)
            field-name = value
            field-name = value
    array-name1 (2)
        field-name = value
        array-name2 (2, 1)
            field-name = value
            field-name = value
        array-name2 (2, 2)
            field-name = value
            field-name = value
Repeating Group (TREE (repeating-group-name1))
    Repeating group-name1
        field-name = value
        field-name = value
```

Figure 6. UDL Default Display Format (Sheet 1 of 2).

```
            repeating group-name2
                field-name = value
                repeating group-name3
                    field-name = value
                repeating group-name3
                    field-name = value
        repeating group-name1
            field-name = value
            field-name = value
            repeating group-name2
                field-name = value
```

Figure 6. UDL Default Display Format (Sheet 2 of 2).

ADAPT I Commands

INTRODUCTION – This paragraph provides a final specification for user commands available under ADAPT I. Generally, commands, as contrasted to statements, are more implementation-dependent and their utility is based on specific users and the hardware/operating system suite under which the system exists. Therefore, as user requirements become more defined and as experience is gained with the ADAPT I/UNIX environment, it is expected that this set of commands will be altered and/or expanded. The dichotomy between

statements and commands discussed here is important since it minimizes the proliferation of UDL "dialects." Commands allow system designers to customize systems for individual users without having to alter more fundamental language constructs (in this case, UDL statements and data structures). This makes the fundamental language constructs "portable" from application to application; i. e., they are standardized.

Following are the syntax and semantics for ADAPT I commands.

ADAPT COMMAND – The ADAPT command provides entry into the ADAPT I system.

Syntax –

<adapt-command> = ADAPT

Semantics - After users have successfully logged onto the TAS system, their terminals are constantly monitored by a TAS command interpreter called the TAS Shell. The ADAPT process is called by the TAS Shell program when users enter the ADAPT command. ADAPT validates that the user is a valid ADAPT user, and if this is the case, the ADAPT I EXEC is called to process the UDL statements/commands. If the user is not a valid ADAPT user, the user is duly informed and returned to the TAS Shell (technically, the ADAPT command is a TAS command, not an ADAPT command).

QUIT COMMAND - The QUIT command provides an exit from the ADAPT I system and returns the user to the TAS environment.

Syntax –

<quit-command> = QUIT;

Semantics – When users choose to leave ADAPT I, they must enter the QUIT command. This command returns the user to the TAS

Shell program. At this time, the user may choose to utilize other TAS subsystems.

Although the user has left ADAPT, the lists generated by him (via the SAVE command) are retained by ADAPT I under the user's identifier. Also, any logical transaction responses which may be received from the network for the user are saved by ADAPT I for later perusal by the user.

OPEN COMMAND – The OPEN command specifies the file(s) that a user wishes to access.

Syntax –

<open-command> = OPEN <file-name-list>;

<file-name-list> = <file-name>, <file-name-list> | <file-name>

Semantics – Users may open one or more files during an ADAPT I session. A file must have been opened prior to its reference via an ADAPT I FIND statement. The primary purpose behind the OPEN command is to establish overall access privileges of the user and the user's terminal with respect to the host where the file resides and to the actual file specified.

CLOSE COMMAND – The CLOSE command closes one or more files from future access.

Syntax –

<close-command> = CLOSE <file-name-list>; | CLOSE;

Semantics – Users may close one or more files during their ADAPT I session by using the CLOSE command. The entry of CLOSE without a file-name-list closes all open files. When users exit (refer to QUIT command), all their open files are automatically closed.

EXECUTE COMMAND – The EXECUTE command allows ADAPT I users to input statement/command sequences from a predefined text file.

Syntax –

```
<execute-command>   =  EXECUTE TRANFILE <execute-clause>;|
                       EXECUTE SCHEMA <execute-clause>;|
                       EXECUTE <execute-clause>;
<execute-clause>    =  <character-constant> ECHO|
                       <character-constant>
```

Semantics – The <character-constant> following EXECUTE is the name of a user-defined text file which contains a series of ADAPT I statement/command sequences. ADAPT I retrieves directives from this file until the file is exhausted and then returns control to the user's terminal. ECHO is an optional specification which, if present, results in the statement/command sequences being echoed back to the terminal. If ECHO is not present, the statement/command sequences are not output. At all times, error diagnostics which may occur are directed to the terminal. If the user suspects errors may be present, it is wise to specify ECHO since, otherwise, the error diagnostics are not associated with the incorrect statements.

MODE COMMAND – The MODE command allows an ADAPT I user to either specify that statement/command sequences are to be processed fully or are to be checked only for syntactic and semantic errors.

Syntax –

<mode-command> = MODE <mode-arg>;

<mode-arg> = VALIDATE | ACTION

Semantics – The MODE command with an argument of VALIDATE requests ADAPT I to accept all following statements and commands for syntactic and semantic validation only. Only a MODE command with an argument of ACTION restores ADAPT to its normal processing state. When a user logs onto ADAPT I, the system is in an action mode. When the system is in the validate mode, no commands are processed, with the exception of the MODE and EXECUTE commands.

VIEW COMMAND – The VIEW command allows a user to display files and file schema definitions defined in ADAPT I, names of lists currently defined for that user, and names of labels used during the current ADAPT session.

Syntax –

<view-command> = VIEW <view-clause>;

<view-clause> = SCHEMA (<file-name>) | LIST | LABEL | FILE | TRANFILE (<file-name>) | TRANS | TRANS (<trans-number>)

<trans-number> = <integer>

Semantics – The VIEW command allows the user to display various types of information concerning his or her transactions via ADAPT, and the file structures defined for ADAPT. Six option-specifiers exist for the view command: SCHEMA, LIST, LABEL, FILE, TRANFILE, and TRANS.

The files defined for ADAPT are listed when a user inputs the VIEW FILE command. With the VIEW SCHEMA command, a user can display a schema for a specific file defined in ADAPT I. This display is quite similar to that of the UDL default format since the file schema is presented as an indented tree structure (if the file contains hierarchical structures). Of course, instead of field values, each field's data type, access attributes and size are provided. The VIEW TRANFILE can be used by the superuser to peruse the transformation data of an ADAPT file. The data, basically the UDL and the host names for the file's aggregates and fields, are displayed in the indented tree format. If the file is positional in nature, the position data and the number of occurrences (if a field is multivalued) are output along with the names of each field.

Users may also view the list-names and labels currently defined for them by specifying VIEW LIST and VIEW LABEL. Users may ascertain the current statuses of their transactions with batch hosts via the VIEW TRANS command. The output from this command contains the transaction numbers, associated filenames and statuses. If the user specifies a transaction number at the end of the VIEW TRANS command, the output is dependent upon the status of the transaction and the contents of the original request. If the status of the transaction is something other than answered (ANSR), an error message is output. Otherwise, if the user had done a SAVE, a hit count is output; if the user had done a DISPLAY, the hit count and the requested data are output for the user's perusal.

SAVE COMMAND – The SAVE command allows users to save actual file records transmitted across the network.

Syntax –

<save-command> = SAVE SOURCE (<tag>) <list-name>;

Semantics – Users may save entire records that were isolated by a previous FIND statement. The tag is a label of a FIND statement, and list-name is a name to be associated with the list of saved records. This name is local for the user and cannot duplicate an existing list-name or label. Lists of records may be displayed in part or in entirety with a UDL DISPLAY statement. List-names associated with the user may be displayed with a VIEW LIST command.

DELETE COMMAND – The DELETE command is used to delete file schemas, tranfiles and user lists.

Syntax –

<delete-command> = DELETE <delete-clause>;

<delete-clause> = SCHEMA (<file-name>) | LIST (<list-name>) | TRANFILE (<file-name>)

Semantics – File schemas and tranfiles that exist in ADAPT I can be removed from the system by the superuser with the DELETE command. The deletion of a file schema also deletes that file's tranfile, if it exists. Lists assigned to the user initiating the DELETE LIST command can be removed from the system, freeing mass storage space and the list-name.

ADAPT I DATA DEFINITION LANGUAGE (DDL)

Introduction

The Data Definition Language (DDL) provides the ADAPT I superuser with the facility to define files for subsequent use with the Uniform Data Language (UDL). All files which are to be referenced with UDL under ADAPT I (although they are resident in the various data bases connected to the network), must have their logical organization be made known to ADAPT I in order for ADAPT I to generate appropriate language transformations. DDL provides a definition of a file in terms of UDL data structures, not in terms of the data structures of the actual target systems.

DDL generates dictionary definitions for each file defined. These definitions can be deleted and/or replaced with new definitions whenever a file's structure is modified at its resident host.

Syntax

<schema-definition>	=	<schema-statement> <schema-clause> ESCHEMA;
<schema-statement>	=	SCHEMA <file-name> <database-type>;
<database-type>	=	LOCAL | REMOTE (BATCH) | REMOTE (INTER)
<schema-clause>	=	<field-def-blk> <aggregate-def-blk> | <field-def-blk>
<aggregate-def-blk>	=	<aggregate-definition> <aggregate-def-blk> | <aggregate-definition>
<aggregate-definition>	=	<array-definition> | <rg-definition>
<field-def-blk>	=	<field-definition> <field-def-blk> | <field-definition>
<field-definition>	=	FIELD <field-name> <field-clause>;

<field-clause>	=	<field-type> <data-type> ATT (<interro-att>, <disp-att>) <size-clause>
<field-type>	=	SV | MV
<data-type>	=	CHAR | NUM | GEO
<interro-att>	=	KEY | NKEY | DEP
<disp-att>	=	VIS | INV
<size-clause>	=	<integer> | V
<array-definition>	=	<array-statement> <field-def-blk> EARRAY;
<array-statement>	=	ARRAY <array-clause>;
<array-clause>	=	<array-name> <array-size> <parent-name> | <array-name> <array-size>
<array-size>	=	<integer>
<parent-name>	=	<array-name> | <repeating-group-name>
<rg-definition>	=	<rg-statement> <field-def-blk> ERGROUP;
<rg-statement>	=	RGROUP <rg-clause>;
<rg-clause>	=	<repeating-group-name> <rg-parent-name> | <repeating-group-name>
<rg-parent-name>	=	<repeating-group-name>

Semantics

For completeness, the foregoing syntax shows DDL schema constructs as a total definitional block. In practice, the schema is produced as a set of ordered DDL statements. A DDL sequence is indicated by the schema-statement. The schema-statement, denoted by SCHEMA, contains a required database-type. The schema-statement contains a file-name (up to eight characters) and the mode of operation of the data base where the file resides The data base is either local (LOCAL) or a distant (REMOTE) target host system, whose mode of operation is either batch (BATCH) or interactive (INTER). The DDL sequence is terminated with an end-schema statement, ESCHEMA.

Following the schema-statement is the schema-clause, a block of data structure definitions making up the file's logical structure. The schema-clause contains two basic types of definitions: those defining fields and those defining aggregates. Aggregate definitions are optional, depending on whether they exist within a file, but a field definition block is mandatory. That is, every file has a basic data set which contains at least one field. The basic data set definition must occur first in the schema-clause. This definition block is formed from a sequence of field-definition-statements. A field definition statement is indicated by the token FIELD followed by the field's name (up to eight characters) and a field-clause. All field names must be unique within a file. The mandatory field-clause specifies the field's type, its data-type, access attributes, and size. Two types of fields exist in UDL, single-valued fields indicated by SV and multivalued fields indicated by MV. Following the specification of the field-type is the data-type specification. Three data-types are recognized in UDL: character, denoted as CHAR; numeric, denoted as NUM; and geographic data type, GEO. As with the field-type, a data-type must be specified. Each field also has associated with it an interrogation attribute and a display attribute. Attribute specifications are indicated by the token ATT. Valid interrogation attributes are KEY, for fields which can be referenced in any FIND statements; NKEY (non-key), for fields which cannot be referenced in FIND statements; and DEP (dependent) for fields which can only be referenced in dependent FIND statements. Two display attributes are recognized in UDL: visible (VIS) and invisible (INV). The maximum field width (as an ASCII representation, regardless of the data type) must be specified for each field. For single-valued and invisible multivalued fields which contain true variable length character strings, such as extensive text, a V may be substituted for the width specification. In most situations, however, the field width should be specified.

If aggregates are present in a file, they must be defined following the basic data set definition block. Aggregates are defined as either arrays or repeating groups. An array definition block begins with an array-statement, denoted by ARRAY. The array-statement specifies the array's name (up to eight characters), its order, and, optionally, its parent's name. The order specifies the maximum number of occurrences allowed for the array. The parent name is the name of another aggregate, either a repeating group or another array. If a parent name is not specified, the array belongs to the basic data set. Following the array-statement is a mandatory field-definition-block. At least one field must exist for an array. Following the field-definition-block is an end-array statement, denoted by EARRAY, terminating the array's definition.

A repeating group definition block begins with a repeating-group-statement, denoted by RGROUP. The repeating-group-statement contains the name of the repeating group (up to eight characters) and an optional parent name. The parent name must be that of another repeating group (arrays cannot be parents of repeating groups). If the parent specification is missing, the repeating group belongs to the basic data set. Following the repeating-group-statement is a mandatory field-definition-block. A repeating group must have at least one field defined for it. The end-repeating-group statement terminates a repeating group definition and is denoted by ERGROUP.

Only hierarchical structures are allowed in records and therefore an aggregate can have at most one parent. Also, an aggregate cannot directly or indirectly (through other aggregates) be its own parent.

ADAPT I TRANSFORMATION DEFINITION LANGUAGE

Introduction

The Transformation Definition Language (TDL) provides the ADAPT I superuser with a mechanism to input host file-specific information required to generate file transformation dictionaries. Each file accessible under ADAPT I must be defined twice: the UDL interpretation of the file must be supplied via the Data Definition Language (DDL) and the transformation-specific information must be supplied via TDL. Although files do not actually exist under ADAPT I (i. e. , there is no local data base manager), information such as the logical file organization, local and target-system nomenclature and file data positioning must be made known to ADAPT I via DDL and TDL in order to generate the appropriate language transformations. It is interesting to note that a local file capability as envisioned for later ADAPT phases requires only the DDL definition, since the file is created and manipulated locally.

The data dictionary definition for a file must exist before the transformation specific information can be processed. Basically, the transformation dictionary stored for a file contains:

a. File name as known on host
b. Data base (host file resides on)
c. Number of characters transmitted per record in a character position dependent file (if applicable)
d. "Key" field (if applicable)
e. Aggregate names as known on host
f. Field names as known on host
    1) Position field description (if applicable)
    2) Number of occurrences (if applicable)

Syntax

<tranfile-definition> = <tranfile-statement> ETRAN; |
<tranfile-statement> <tranfile-block> ETRAN;

<tranfile-statement> = TRANFILE <tranfile-clause> <file-definition>;

<tranfile-clause> = <file-name> | <file-name> (<character-constant>)

<file-definition> = <database-clause> | <database-clause> <pos-clause> | <database-clause> <key-clause> | <database-clause> <pos-clause> <key-clause>

<database-clause> = DATABASE (<database-id>)

<pos-clause> = POSITION (<integer>)

<key-clause> = KEY (<field-name>)

<tranfile-block> = <tran-field-state> <tranfile-block> | <tran-agg-state> <tranfile-block> | <tran-field-state> | <tran-agg-state>

<tran-agg-state> = <tran-array-state> | <tran-rg-state>

<tran-field-state> = FIELD <field-clause> <field-pos-clause>; | FIELD <field-clause>;

<field-clause> = <field-name> (<character-constant>) | <field-name> (<character-constant>, <character-constant>) | <field-name>

<field-pos-clause> = POS (<integer>) | POS (<integer>, <integer>)

<database-id> = RYETIP | DIAOLS | SOLIS | ISSPIC

<tran-array-state> = ARRAY <array-name> (<character-constant>);

<tran-rg-state> = RGROUP <repeating-group-name> (<character-constant>);

Semantics

As with the Data Definition Language, the foregoing syntax shows the TDL transformation constructs as a total definitional block. In actuality, the tranfile is a set of ordered TDL statements, initialized by the tranfile-statement and terminated by the end-tranfile-statement.

The tranfile statement, designated by TRANFILE, consists of 1) the file-name as recognized by the target host system, 2) the host database where the file resides, 3) the number of characters transmitted per record and, 4) the "key" field. The specified file-name is the UDL file-name associated (via DDL) with a particular file structure and set of data dictionary files. If the file is known by a different name by the target system, that name follows in the form of a character-constant. The statement also contains a mandatory database identifier and optional maximum record size specifier and key field-name. The database identifier, denoted by DATABASE, indicates the target host system where the file actually resides. The number of characters transmitted per record in a character-position dependent file is input as an integer after the keyword POSITION. Lastly, the key-clause consists of the keyword KEY followed by the name of the key field, a previously defined field for the file. A key field, when applicable, is required for the interpretation of potential duplicate records in response to multiple queries, and allows the output translator to determine when the output for each record begins. Following the tranfile statement, an optional group of statements, referred to as a tranfile-block, may appear. The tranfile-block consists of a series of transformation aggregate and field definitions. The order of their appearance need not reflect the order of their input for the file's data dictionary files.

When an aggregate is known by a name other than the locally assigned UDL aggregate name, the target host system name is specified in the tran-agg-state. The tran-agg-state consists of three parts: 1) the keyword, either ARRAY or RGROUP, which designates the structural design of the aggregate, 2) the name (array-name or repeating-group-name) which is the UDL name for the aggregate and, 3) the character-constant which is the name by which the target host system recognizes the aggregate.

The transformation field statement has several constructs, depending on the situation and the required information. If the file is not position oriented, the local UDL field-name is followed by one or two character-constants containing the name(s) by which the field is known by the target host system. If, however, the file is position dependent, the tran-field-state consists of a field-name, optionally followed by one or two character-constants (target host system name(s)), and the position keyword POS, which, in turn, is followed by one or two integers. The first integer is the character position, i. e. where the field's data is stored relative to the beginning of the record. The field position information of each field within a position-dependent file must be specified within the tranfile-block. The second optional integer in the field-pos-clause indicates the number of occurrences associated with the field and is required if the field is a multi-valued field.

A tranfile definition is terminated by an end-tranfile statement, ETRAN.

CROSS REFERENCE FOR ADAPT I ERROR DIAGNOSTICS AND MESSAGES

This section provides a listing of all error diagnostics or messages produced by ADAPT I statements and commands. For each diagnostic the following information is given: the reference number, the text, the probable cause of the error, the action to be taken if any, and a list of ADAPT statements and commands which produce the diagnostic/message.

ADAPT ERROR 1	ILLEGAL FIELD DESCRIPTION FOR ***
From:	FIELD (DDL)
Cause:	The size of the field which you are defining via DDL is too large.
Action:	Retype the statement with a smaller field size.
ADAPT ERROR 2	PARENT OF REPEATING GROUP *** CANNOT BE AN ARRAY
From:	ESCHEMA (DDL)
Cause:	The repeating group which you are defining via DDL cannot have an array for a parent.
Action:	Review the file's structure and determine the correct parent of the repeating group.
ADAPT ERROR 3	DUPLICATE NAME ***
From:	RGROUP, ARRAY, and FIELD (DDL)
Cause:	Field and aggregate names must be unique for a given file's data definition.
Action:	Retype the statement using a unique name.
ADAPT ERROR 4	INVALID ATTRIBUTES (NKEY AND INV) FOR FIELD ***
From:	FIELD (DDL)
Cause:	A field cannot be assigned simultaneously the attributes invisible (INV) and non-keyed (NKEY).
Action:	Review the attributes of the field and adjust the field statement accordingly.

ADAPT ERROR 5 — ATTRIBUTES OF FIELD *** DO NOT ALLOW VARIABLE LENGTH

From: FIELD (DDL)

Cause: A field's data cannot be variable in length if the field is visible and multivalued. In the following situations, a variable length attribute is valid:

a. Field is invisible and multivalued.
b. Field is single-valued.

Action: First review the characteristics of the field and then adjust the field statement.

ADAPT ERROR 6 — TOO MANY FIELDS

From: FIELD (DDL)

Cause: You have exceeded the maximum number of fields that can be assigned to a file.

Action: Review the file structure and act accordingly by removing the least important fields.

ADAPT ERROR 7 — TOO MANY AGGREGATES

From: ARRAY, RGROUP (DDL)

Cause: You have exceeded the maximum number of aggregates that can be defined for a file.

Action: Review the file structure and act accordingly by removing the least important aggregates.

ADAPT ERROR 8 — UNEXPECTED END-AGGREGATE STATEMENT

From: ERGROUP, EARRAY (DDL)

Cause: An EARRAY or an ERGROUP statement is misplaced. A properly used end-aggregate-statement should appear after either an array or a repeating group definition.

Action: Remove the statement from the present sequence of statements.

ADAPT ERROR 9 — END-AGGREGATE STATEMENT DOES NOT MATCH AGGREGATE STATEMENT

From: EARRAY, ERGROUP (DDL)
Cause: An EARRAY statement terminates the definition of an array. An ERGROUP statement terminates the definition of a repeating group.
Action: Choose the correct end-aggregate statement.

ADAPT ERROR 10 — AGGREGATE *** HAS NO FIELDS DEFINED FOR IT

From: EARRAY, ERGROUP (DDL)
Cause: At least one field must be defined for each aggregate. This rule also applies to the basic data set. See error 16.
Action: Define at least one field for the aggregate.

ADAPT ERROR 11 — AGGREGATE *** IS UNDEFINED

From: ESCHEMA (DDL)
Cause: The specified aggregate has been specified as the parent of another aggregate but has never been formally defined.
Action: Either define the aggregate or reenter the statement which specified it as a parent.

ADAPT ERROR 12 — CIRCULAR RELATIONSHIP FOR AGGREGATE ***

From: ESCHEMA (DDL)
Cause: The specified aggregate is involved in an invalid circular relationship: the given aggregate is the parent of an aggregate which, if its parent-son link is followed, is essentially the parent of the given aggregate.
Action:
1. Review the file structure and redefine it accordingly.
2. Review the DDL definition of the file structure and correct its inconsistencies.

ADAPT ERROR 13 NO PARENT AGGREGATE FOR FIELD ***

From: FIELD (DDL)
Cause: The specified field does not have either an array or repeating group with which it is to be associated. Once the basic data set has been defined, a FIELD statement must appear between an aggregate statement (ARRAY or RGROUP) and an end-aggregate statement (EARRAY or ERGROUP).
Action: The FIELD statement is to be reinitiated or removed from the file's DDL definition.

ADAPT ERROR 14 ARRAY *** DIMENSION IS OUT OF RANGE

From: ARRAY (DDL)
Cause: An array's dimension cannot be less than one (1) or greater than four hundred (400).
Action: Reenter the array's dimension within the allowed range.

ADAPT ERROR 15 SIZE OF FIELD DATA HAS EXCEEDED LIMIT FOR AGGREGATE ***

From: FIELD (DDL)
Cause: The maximum size for an aggregate is 5000 bytes. An aggregate's size is equal to the sum of the sizes of the visible fields belonging to the aggregate.
Action: Check the file's data definition for accuracy, and review the file structure. The sizes of the fields or the number of fields defined for the aggregate might have to be adjusted.

ADAPT ERROR 16 NO FIELDS DEFINED FOR BASIC DATA SET

From: ESCHEMA, ARRAY, RGROUP (DDL)
Cause: At least one field must be defined for the basic data set.
Action: Define a field for the basic data set before defining any aggregates.

ADAPT ERROR 17	TOO MANY LEVELS OF ARRAYS FOR AGGREGATE ***
From:	ARRAY (DDL)
Cause:	An array may have at most two levels of arrays to which it is subordinate.
Action:	Redefine the file structure.
ADAPT ERROR 18	DUPLICATE FILENAME ***
From:	SCHEMA
Cause:	Each file defined for ADAPT I must have a unique name.
Action:	Perform a VIEW FILE for a listing of all files for which data definitions exist. Given the established file names, choose a unique file name.
ADAPT ERROR 19	YOU ARE NOT AN ESTABLISHED ADAPT USER. CONTACT THE TASMASTER TO BECOME ESTABLISHED AS AN ADAPT USER
From:	ADAPT
Cause:	The ADAPT command is restricted to those users who are authorized to use ADAPT by the TASMASTER.
Action:	Ask the TASMASTER to establish you as an ADAPT user.
ADAPT ERROR 21	CANNOT DELETE SCHEMA/TRANFILE AT THIS TIME BECAUSE ADAPT USERS ARE LOGGED ON
From:	DELETE SCHEMA, DELETE TRANFILE
Cause:	Due to interdependency of ADAPT activities on the SCHEMA and TRANFILE files, neither can be deleted until all other users are logged off of ADAPT.
Action:	Wait until all other users are logged off of ADAPT.

ADAPT ERROR 22 — THIS STATEMENT/COMMAND IS RESTRICTED TO THE SUPERUSER

From: SCHEMA, TRANFILE, DELETE SCHEMA, DELETE TRANFILE, VIEW TRANFILE, EXECUTE SCHEMA, EXECUTE TRANFILE

Cause: Statements and commands which maintain the ADAPT environment are restricted to the superuser.

ADAPT ERROR 23 — TOO MANY FILES

From: ESCHEMA

Cause: You have exceeded the maximum number of target system files allowed for ADAPT I.

ADAPT ERROR 24 — NESTED EXECUTE COMMANDS ARE NOT ALLOWED

From: EXECUTE SCHEMA, EXECUTE TRANFILE, EXECUTE

Cause: An execute statement cannot appear within a file which has been named within a previous execute statement as an input file.

Action: Remove the execute statement from the text file.

ADAPT ERROR 26 — INVALID DATABASE ***

From: TRANFILE

Cause: Valid database names for ADAPT I are RYETIP, DIAOLS, SOLIS and ISSPIC.

Action: Retype the TRANFILE statement.

ADAPT ERROR 27 — NO END-AGGREGATE STATEMENT FOR AGGREGATE ***

From: RGROUP, ARRAY, ESCHEMA (DDL)

Cause: DDL requires that the data definition of either an array or repeating group is terminated by an end-statement, either EARRAY or ERGROUP.

Action: Insert the appropriate end statement after the definition of each aggregate.

ADAPT ERROR 28 — UNEXPECTED EOF IN A SCHEMA OR TRANFILE DEFINITION

From: N/A
Cause: The input from an execute file has terminated before either an ESCHEMA or ETRAN statement has been processed.
Action: Add the appropriate end-statement to the end of the text file.

ADAPT ERROR 29 — DATA INCONSISTENCY IN TRANSACTION COUNTS

From: DISPLAY, SAVE
Cause: Error 29 is an internal ADAPT error.
Action: Please take notes recreating the series of events leading to the error and submit your notes to the TASMASTER.

ADAPT ERROR 30 — CANNOT RECOGNIZE THE TARGET SYSTEM DATA. THESE RECORDS CANNOT BE PROCESSED.

From: VIEW TRANS
Cause: The process which scans the target system records does not recognize the contents of the record.
Action: Notify the TASMASTER.

ADAPT ERROR 31 — TAG *** IS INVALID

From: FIND, SAVE, DISPLAY
Cause: The tag specified after the keyword SOURCE is not a valid label name. The tag has not been a label of one of your previous FIND statements.
Action: List the current labels via the VIEW LABEL command.

ADAPT ERROR 32 — LIST NAME *** IS INVALID

From: DISPLAY, DELETE LIST
Cause: The specified list name is not found in the ADAPT files.
Action: Perform a VIEW LIST for a display of your current list names.

ADAPT ERROR 33 THIS TRANSACTION IS NOT COMPLETED

From: VIEW TRANS
Cause: The transaction which you are attempting to view is not completed and cannot be viewed.
Action: Wait until the transaction is completed. Perform a VIEW TRANS to output the status of your transactions.

ADAPT ERROR 34 FILE NAME *** IS INVALID

From: OPEN, CLOSE, FIND, VIEW SCHEMA, VIEW TRANFILE
Cause: The data definition for the specified file does not exist.
Action: Perform a VIEW FILE for the display of all current ADAPT files.

ADAPT ERROR 35 AGGREGATE NAME *** IS INVALID

From: FIND, ARRAY (TDL), RGROUP (TDL)
Cause: The specified aggregate is not a defined array or repeating-group for the file.
Action: Perform a VIEW SCHEMA for the complete listing of the particular file's data structure.

ADAPT ERROR 36 FIELD NAME *** IS INVALID

From: FIND, FIELD (TDL)
Cause: The specified field is not a defined field for the file.
Action: Perform a VIEW SCHEMA for a complete listing of the specific file's data structure.

ADAPT ERROR 37 FIELD *** MUST BE VISIBLE TO BE DISPLAYED

From: DISPLAY
Cause: A field which is assigned the attribute of invisible cannot be specified in a DISPLAY statement.
Action: Reword and retype the DISPLAY statement.

ADAPT ERROR 38 CANNOT REFERENCE DEPENDENT TAG ***

From: FIND

Cause: When a label of a FIND statement is used as a reference tag for a second FIND statement, the label for the second FIND statement is a dependent label. This latter label cannot be used as a tag in yet another FIND statement. This would imply a second level of dependency in the FIND statement.

Action: Reevaluate the FIND statement and its relationship with previous FIND statements.

ADAPT ERROR 39 CANNOT REFERENCE TAG *** BECAUSE IT WILL BE REPLACED BY A NEW TAG

From: FIND

Cause: The most current labels for a user's interrogations are maintained for that user. Once the maximum number of labels have been used, the oldest label is always replaced by the newest label. If the FIND statement is dependent on the label about to be deleted, error 39 is output and the processing of the FIND statement is terminated.

Action: Retype the FIND statement upon which the rejected FIND statement is dependent. Once this statement has been processed, retype the dependent FIND statement.

ADAPT ERROR 40 DUPLICATE LIST OR LABEL NAME – ***

From: SAVE, FIND

Cause: A list name or a label (tag) must be a unique name, differing from a user's current list of label and list names.

Action: Retype the statement with a unique list name or label. Perform a VIEW LIST and/or VIEW LABEL for a complete display of your current listnames or labels.

ADAPT ERROR 41 — FIELD *** MUST BE KEYED TO BE USED IN AN INDEPENDENT FIND STATEMENT

From: FIND
Cause: An independent FIND statement requires all specified fields to be keyed.
Action: Retype the statement. Create two FIND statements: the first is independent and the second is dependent with the specified field in this statement.

ADAPT ERROR 42 — FIELD *** MUST BE KEYED OR DEPENDENT TO BE USED IN A DEPENDENT FIND STATEMENT

From: FIND
Cause: The fields in a dependent FIND statement must have the attributes of keyed or dependent. Non-keyed fields cannot be used in FIND statements.
Action: Retype the FIND statement omitting the field name within the original FIND statement.

ADAPT ERROR 43 — INCORRECT CONSTANT TYPE FOR FIELD ***

From: FIND
Cause: Three data types exist for fields: character, numeric and geographic. The constant type which is being specified for the field name and operator must be analagous to the field's data type.
Action: VIEW SCHEMA displays a file's structure including each field, its name, data type, size and attributes.

ADAPT ERROR 44 — AGGREGATE *** MUST BE A REPEATING GROUP

From: FIND
Cause: Aggregates which are used as scope qualifiers must be repeating groups.
Action: Review the field's structure and retype the statement.

ADAPT ERROR 45 — FIELD *** MUST BE IN THE SCOPE REPEATING GROUP

From: FIND

Cause: The fields specified within a repeating group scope qualifier must be defined for that repeating group.

Action: Review the file's structure via the VIEW SCHEMA command and retype the statement.

ADAPT ERROR 46 — INVALID SUBSCRIPT FOR FIELD ***

From: FIND, DISPLAY

Cause: The subscripts for a field must not exceed the maximum size (dimension) assigned to the array to which the field is subordinate. The subscript to the farthest right corresponds to the array which is the direct parent of the field. As you proceed left, the subscript refers to the array which is the parent of the array whose subscript is to its right.

Action: Review the file structure via VIEW SCHEMA to list each array and its maximum dimension and those fields defined for each array.

ADAPT ERROR 47 — FIELD *** HAS TOO MANY SUBSCRIPTS

From: FIND

Cause: The number of subscripts in a field's dimension must equal the number of levels of arrays to which the field is subordinate.

Action: Review the file's structure via VIEW SCHEMA and adjust the FIND statement accordingly.

ADAPT ERROR 48 — A FIND STATEMENT MUST HAVE A LABEL

From: FIND

Cause: Each FIND statement must have a label; otherwise, the FIND statement is not processed.

Action: Retype the statement with a label at the beginning.

ADAPT ERROR 49 FIELD *** HAS TOO FEW SUBSCRIPTS

From: FIND
Cause: The number of subscripts in a field's dimension must equal the number of levels of arrays to which the field is subordinate.
Action: Review the file's structure via VIEW SCHEMA and adjust the FIND statement accordingly.

ADAPT ERROR 50 FIELD NAME OR AGGREGATE NAME *** IS INVALID

From: DISPLAY
Cause: The specified name is neither a field nor an aggregate defined for the file.
Action: Review the file's structure via VIEW SCHEMA and retype the DISPLAY statement accordingly.

ADAPT ERROR 51 FIELD *** MUST HAVE SUBSCRIPTS IN A FIND STATEMENT

From: FIND
Cause: The parent of the specified field is an array, implying the need for the field to be subscripted. A field does not necessarily have to be subscripted for DISPLAY statements.
Action: Review the file's structure via VIEW SCHEMA and adjust the FIND statement accordingly.

ADAPT ERROR 52 TRANSACTION NUMBER *** IS INVALID

From: VIEW TRANS
Cause: The transaction number you typed in is not assigned to any of your current queries/responses.
Action: Perform a VIEW TRANS for a complete listing of your transactions and their statuses. The entire three digits must be entered.

ADAPT ERROR 53 TAG OR LISTNAME *** IS INVALID

From: DISPLAY
Cause: The specified name is not a current label or list name in your ADAPT files.
Action: VIEW LIST and VIEW LABEL give you a current listing of valid list and label names.

ADAPT ERROR 54 USER INPUT FILE *** IS INVALID

From: EXECUTE, EXECUTE SCHEMA, EXECUTE TRANFILE

Cause: The input file as specified in the execute statement is not an existing text file.

Action: The TAS command LIST outputs a listing of your text files.

ADAPT ERROR 55 STATEMENT TOO LONG

From: Any overlength statement.

Cause: The ADAPT command/statement exceeds the maximum statement length.

Action: Review the format of the command/statement and retype.

ADAPT ERROR 56 INVALID GEOGRAPHIC EXPRESSION FOR FIELD ***

From: FIND

Cause: UDL has three geographic operators. Each operator accepts only specific geographic expressions. The chart below indicates acceptable relationships.

Geographic Operator	Geographic Expression		
	Circle	Poly	Route
Inside	x	x	
Outside		x	
Along			x

Action: Reword your FIND statement, matching the correct geographic expression with the geographic operator.

ADAPT ERROR 57	TOO MANY/FEW GEOGRAPHIC CONSTANTS FOR FIELD ***
From:	FIND
Cause:	The geographic expressions POLY and ROUTE both require at least 3 and at most 506 geographic constants.
Action:	Review the geographic expression format and retype the FIND statement.
ADAPT ERROR 58	RADIUS OR BAND WIDTH MUST BE OF RANGE 0. 1 to 2000 FOR FIELD ***
From:	FIND
Cause:	The band width for the ROUTE geographic expression and the radius for the CIRCLE geographic expression must be a number within the range of 0. 1 and 2000 inclusively.
Action:	Reenter the statement with a correct value.
ADAPT ERROR 59	DEGREES, MINUTES, OR SECONDS IS OUT OF RANGE FOR A GEOGRAPHIC CONSTANT FOR FIELD ***
From:	FIND
Cause:	Each geographic constant contains a latitude and longitude expression in that order. Both latitude and longitude have four parts in their full expressions: degrees, minutes, seconds, and direction (N,S, E, W). If the minute and seconds are not present, the default values are 0. The chart below gives the allowable numeric ranges for the degrees, minutes and seconds for both latitude and longitude.

	Numeric Value	
	Low	High
latitude degrees	0	90
latitude minutes	0	59
latitude seconds	0	59
longitude degrees	0	180
longitude minutes	0	59
longitude seconds	0	59

Action: Review the valid formats for geographic constants and retype the FIND statement.

ADAPT ERROR 60 WRG, ORG OPERATORS REQUIRE NUMBERS TO BE IN INCREASING NUMERIC ORDER

From: FIND

Cause: The operators WRG (within range) and ORG (out-of-range) require two numeric expressions where the first number must be less than the second number.

Action: Reverse the two numbers.

ADAPT ERROR 61 INVALID NUMBER OF SCHEMA/TRANFILE STATEMENTS

From: TRANFILE, SCHEMA

Cause: For each SCHEMA/TRANFILE definition at most one SCHEMA/TRANFILE statement may appear within the definition. The presence of another such statement nullifies the definition.

Action: Remove the statement.

ADAPT ERROR 62 TOO MANY UNIQUE BOOLEAN VARIABLES

From: FIND

Cause: A FIND statement may possess at most eight (8) unique variables where a variable is defined to consist of three parts. The first part is a field name, the second part is the operator, and the last part is the expression (numeric, geographic, or character); e. g., if two variables have identical field names and operators but the expressions differ, the variables are counted as two (2) unique variables.

Action: Reduce the number of unique variables in your FIND statement, or retype the FIND statement as two statements, an independent and a dependent FIND (if possible).

ADAPT ERROR 63 TOO MANY LISTS

From: SAVE

Cause: You have exceeded the maximum number of lists allowed.

Action: Delete an old list which is no longer useful.

ADAPT ERROR 64 TRANSFORMED STATEMENT TOO LONG FOR TARGET SYSTEM

From: FIND, DISPLAY, SAVE

Cause: Each target system has a maximum length for a statement which it can accept and process.

Action: Reduce the complexity of your FIND statement.

ADAPT ERROR 65 TOO MANY RECORDS REQUESTED WITH THIS STATEMENT

From: FIND

Cause: If after the interrogation (FIND) statement has been logically evaluated and it has been determined that you are actually requesting all data stored in the target system file, this error is output; e. g., LAB FIND IN TESTFILE VAR GT 42 OR NOT (VAR GT 42); is actually asking for all records in TESTFILE.

Action: Evaluate your statement and pinpoint the problem. Reword the FIND statement, ensuring that you are requesting a subset of the file.

ADAPT ERROR 66 — CANNOT DELETE LIST ***. IT HAS NOT BEEN DELIVERED.

From: DELETE LIST
Cause: A response to a query must be displayed (i. e., delivered) before the associated list name can be deleted.
Action: Display the list's data and then delete it.

ADAPT ERROR 67 — CONSTANT IS INCORRECTLY FORMATTED

From: Any statement
Cause: The construction of a constant is incorrect.
Action: Study the statement's valid format and valid constant/expression formats.

ADAPT ERROR 69 — NO RECORDS WOULD BE REQUESTED WITH THIS FIND STATEMENT

From: FIND
Cause: This error message corresponds closely with error 65. Having logically evaluated the interrogation FIND statement it is determined that the user is not requesting any data.
Example:
LABEL FIND IN TESTFILE VAR GT 42 AND NOT (VAR GT 42)
Since no record could possibly contain a value of VAR which is both greater and not greater than 42 simultaneously, the user is not requesting any data.
Action: Analyze your data and locate the error. Now retype your FIND statement.

ADAPT ERROR 70 — TOO MANY CHARACTERS (FOR NAME OR CONSTANT) FOR FIELD ***

From: Any statement

Cause: One of the following three reasons caused this error: (1) In a FIND statement, the character constant contains more characters than the field size allows as defined via DDL. The field's size limits the number of characters upon which a search map can be attempted. (2) In a TRANFILE statement the number of characters in the file's name cannot exceed 10 characters. (3) The maximum number of characters per name is 8; per character constant is 200; per number is 12.

ADAPT ERROR 73 — TRANSACTION FOR LIST *** IS NOT COMPLETE

From: VIEW TRANS

Cause: Before the response to an interrogation can be perused by a user, the transaction between the target system and ADAPT must be completely finished.

Action: Wait. The VIEW LIST command provides the means to list the status of your lists.

ADAPT ERROR 75 — FILE *** NOT OPEN

From: CLOSE, FIND, DISPLAY

Cause: One of the following three reasons caused this error:

1. You are closing a file which has not been opened.
2. You are interrogating a file which has not been opened.
3. You are displaying contents of an unopened file via a DISPLAY statement.

Action: Open the file.

ADAPT ERROR 76 — FILE *** ALREADY OPENED

From: OPEN

Cause: The file has previously been opened.

Action: None.

ADAPT ERROR 77 TOO MANY OPEN FILES

From: OPEN
Cause: You have exceeded the maximum number of target system files which may be opened at any given moment.
Action: Close some of the open files.

ADAPT ERROR 78 TRANFILE DATA ALREADY PRESENT FOR FILE ***

From: TRANFILE
Cause: The superuser can redefine a file's TRANFILE definition only after he has deleted the TRANFILE data via the DELETE TRANFILE statement.
Action: Delete the TRANFILE definition for the file.

ADAPT ERROR 79 INVALID POSITION VALUE FOR FIELD ***

From: FIELD (TDL)
Cause: The position value for a field must be an integer greater than 0 and less than the total number of characters in the file.
Action: Reevaluate the field's position in the file.

ADAPT ERROR 80 INVALID OCCURRENCE COUNT FOR FIELD ***

From: FIELD (TDL)
Cause: A field which is multivalued has an occurrence count which must be an integer greater than 0 and less than 496.
Action: Analyze the number of occurrences for the field and retype the field statement.

ADAPT ERROR 81 POSITION DATA NECESSARY FOR FIELD ***

From: FIELD (TDL)
Cause: Each field in a position-type file must have a position value associated with it. This error is output when the position data is missing.
Action: Determine the field's position value.

ADAPT ERROR 82 LOCAL DATA BASE MANAGER NOT IMPLEMENTED YET

From: DDL
Cause: Self-explanatory.
Action: None.

ADAPT ERROR 83 FIELD *** IS NOT MULTI-VALUED

From: FIELD (TDL)
Cause: Those fields defined to be multivalued in the field's data definition must be assigned an occurrence count in the file's TRANFILE definition for position-type files. Only such fields may have occurrence counts.
Action: Do not assign the indicated field an occurrence count.

ADAPT ERROR 84 OCCURRENCE COUNT NECESSARY FOR FIELD ***

From: FIELD (TDL)
Cause: Those fields specified as multivalued in the field's data definition must be assigned an occurrence count in the file's TRANFILE definition for position-type files.
Action: Assign an occurrence count to the indicated field.

ADAPT ERROR 85 FIELD *** POSITION DATA INVALID FOR THIS FILE

From: FIELD (TDL)
Cause: The file is not a position file; therefore the fields within the file should not be assigned a position value.
Action: Either remove all position values from fields in this file's TRANFILE definition or specify that the file is position oriented in the TRANFILE statement for this definition.

ADAPT ERROR 86 *** HAS BEEN PREVIOUSLY DEFINED

From: FIELD, ARRAY, RGROUP (TDL)
Cause: Either a field or aggregate has already been defined within the TRANFILE's definition.
Action: Remove duplicate definitions.

ADAPT ERROR 87 DISPLAY LIST MUST INCLUDE AN ELEMENT NAME

From: DISPLAY
Cause: The UDL DISPLAY statement must have at least one element in the display list. This implies that a display list must consist of more than character constant strings.
Action: Study the format of the DISPLAY statement and retype the statement.

ADAPT ERROR 88 TRANFILE DATA DOES NOT EXIST FOR FILE ***

From: OPEN
Cause: A file cannot be opened until both the data definition and the TRANFILE definition exist.
Action: Inform the superuser of the missing TRANFILE definition for the indicated file.

ADAPT ERROR 91 TOO MANY LOGICAL TRANSACTIONS

From: DISPLAY, SAVE
Cause: You have exceeded the maximum number of transactions.
Action: Perform a VIEW TRANS in order to determine which previous transactions should be eliminated.

ADAPT ERROR 93 — LOGICAL EXPRESSION WITHIN SCOPE EXPRESSION HAS AN ERROR

From: FIND

Cause: The selection criteria of a scoped expression cannot ask for all or for none of the scoped data. Errors 69 and 65 are closely related to this problem.
Example:
Scope expression which logically asks for all data:

```
RNAME (FLD EQ 5 OR NOT (FLD EQ 5) )
```

Scope expression which logically requests nothing:

```
RNAME (FLD EQ 5 AND NOT (FLD EQ 5) )
```

Action: Analyze the logical expressions in the scope expression and rewrite the statement.

ADAPT ERROR 98 — YOU MUST USE SEPARATE EXECUTE COMMANDS FOR SCHEMA/TRANFILE

From: SCHEMA, TRANFILE

Cause: The file specified in an EXECUTE statement must contain only UDL statements, not DDL or TDL statements.

Action: Use an EXECUTE SCHEMA or EXECUTE TRANFILE statement to process a schema or tranfile definition which is stored in a file.

ADAPT ERROR 100 — REFERENCED FIND STATEMENT HAS AN INVALID HIT COUNT

From: DISPLAY (Interactive only)

Cause: You are trying to display a statement with either a hit count of zero or a hit count greater than fifty.

Action: Redefine the FIND statement and resubmit your query.

ADAPT ERROR 102 — INTERACTIVE HOST DATA CANNOT BE SAVED

From: SAVE (Interactive only)

Cause: The results of an interactive session cannot be saved for later perusing.

Action: None.

ADAPT ERROR 116 INVALID POSITION VALUE

From:	TRANFILE
Cause:	The TRANFILE position value is either zero or beyond the host system's maximum position value.
Action:	Keep the position value between one and 495, inclusively.

CROSS REFERENCE FOR ADAPT I MESSAGES

A message is output by ADAPT either to inform the user of the status of the system, the result of his statement/command, or to instruct the user of his next action. The message text and a list of the ADAPT statements and commands which produce the message are given for each message; however, since most messages are self-explanatory, the cause of the message and any action to be taken is given only if they are necessary. For your convenience, the messages have been alphabetized.

COMMAND COMPLETED

From: QUIT

FILE *** HAS BEEN OPENED

From: OPEN

LIST *** HAS BEEN DELETED

From: DELETE LIST

NO HIT COUNT FOR A BATCH TRANSACTION

From: FIND (Batch only)
Cause: A FIND statement for a batch file is not actually sent to the appropriate host until a SAVE or DISPLAY is input referencing the FIND. Therefore, no hit count is known for the query.

NUMBER OF RECORDS FOUND NOT SAME AS NUMBER SPECIFIED IN OUTPUT

From: VIEW TRANS
Cause: Although the host indicated a specific number of records which satisfied the query, only a subset was actually transmitted. This is conceivably due to the host truncating an oversized output.
Action: Resubmit the query with a refined set of selection criteria.

NUMBER OF RECORDS SELECTED IS ***

From: FIND (interactive), VIEW TRANS

SCHEMA FOR FILE *** HAS BEEN DELETED

From: DELETE SCHEMA

SORRY, HOST DID NOT RECOGNIZE YOUR STATEMENT

From: FIND
Cause: This is an internal ADAPT error.

SORRY, REFERENCED FILE NOT AVAILABLE FROM THE HOST AT THIS TIME

From:	FIND statement to SOLIS
Action:	Reference another file for SOLIS.

THERE ARE NO FILE DEFINED

From:	VIEW FILE
Cause:	No files defined for ADAPT at this time.

TRANFILE FOR FILE *** HAS BEEN DELETED

From:	DELETE TRANFILE

TRANSACTION DATA FOR THIS LIST WAS NOT FOUND, THE LIST IS DELETED

From:	DISPLAY
Cause:	An internal problem has caused this error. The list has accidently been deleted.
Action:	Please take notes on the sequence of statements which lead to the error and give the notes to the TASMASTER.

TRANSACTION NUMBER FOR THIS BATCH QUERY IS ***

From:	DISPLAY, SAVE
Cause:	Your batch query has been assigned a transaction number which is used in VIEW TRANS statements to monitor the status of and to view the response to the transaction.

TRANSACTION NUMBER *** HAS BEEN DELETED

From:	VIEW TRANS

UNRECOGNIZED ERROR MESSAGE AS FOLLOWS:

From:	VIEW TRANS
Cause:	The target host system has sent an error message which ADAPT is not capable of recognizing.

YOU HAVE AUTOMATICALLY BEEN REMOVED FROM THE ADAPT SUB-SYSTEM

Cause: An internal ADAPT system error has caused you to be logged off.

YOU HAVE NO BATCH TRANSACTIONS

From: VIEW TRANS

YOU HAVE NO LISTS SAVED

From: VIEW LIST

YOU HAVE NO LABELS

From: VIEW LABEL

YOUR TRANSACTION HAS BEEN EITHER LOGGED OUT OR LOST, PLEASE RESEND IT

From: VIEW TRANS
Action: Resubmit the original query.

------TO GET THE NEXT SCREEN, HIT THE <NEWLINE> KEY---------

From: Any statement which causes output to be paged to the terminal screen.

APPENDIX A

UNIFORM DATA LANGUAGE (UDL)

A. 1 INTRODUCTION

This section provides a specification for the Uniform Data Language (UDL). The information which follows is divided into three basic groups:

a. Basic definitions.

b. Data structures.

c. UDL statements.

Basic definitions provides descriptions of the character set, the construction of names and labels, and the syntax and semantics of basic expressions. Data structures are described with respect to their logical structure, designated data types, and assigned data attributes. All UDL statements are described where their syntax and semantics are provided in detail.

A. 2 BASIC DEFINITIONS

This paragraph presents the fundamental definitions for the UDL. Included are the character set, data name, and label definitions, and descriptions of the data constants and fundamental expressions. These definitions and their syntax, as well as semantics, will be referenced constantly throughout the remainder of the UDL description.

A. 2. 1 Character Set

The character set assumed by UDL will be the American Standard Code for Information Interchange (ASCII). Out of this character set, the following delimiters, digits, and letters are valid for UDL statement construction.

A. 2. 1. 1 Syntax.

<delimiter> = + | - | / | * | @ | (|) | , | . | = | Δ | ? | % | ; | ' | CR

<digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A. 2. 1. 2 Semantics. The meaning of each of the delimiters is as follows:

+: binary add operator; unary plus operator.

-: binary subtract operator; unary minus operator.

/: division operator; declare delimiter; geographic constant delimiter

*: multiply operator.

@: initial character for variable name.

(: expression, list, or grammatical enclosure (initial).

): expression, list, or grammatical enclosure (terminal).

, : list separator.

. : decimal point; label delimiter.

=: assignment statement delimiter.

Δ: designates a space.

?: mask character for CONTAINS operator.

%: denotes geographic constant.

; : terminates statements and commands

': delimits (initial and terminal) character constant.

CR: new line.

Decimal digits are used to express numeric constants, to form labels and names (except as the first character), and to form character and geographic constants.

A letter may be any letter of the alphabet. Letters may be used in labels, names, and in character and geographic constants.

A. 2. 2 Names and Labels

Names and labels are UDL identifiers which are constructed by the user. Following are the syntax and semantics of UDL names and labels.

A. 2. 2. 1 Syntax.

<name> = <name> <letter> | <name> <digit> | <letter>
<label> = <name> · <label> | <name>
<tag> = <label>

A. 2. 2. 2 Semantics. Names can consist of up to eight characters, where these characters must be either letters or digits. The first character must be a letter. Names are used to identify various entities in a UDL procedure or statement complex. All syntactically correct names are valid constructs in UDL except for those occurring in the reserved word list in table A-1.

Names can be utilized in two basic ways: pure names or as parts of other identifiers. Pure names can occur as a procedure name, a list name, or as a data base structure name (i. e. , file-name, field-name, etc.). Names can also be used to form labels or variable names.

Labels are user-defined identifiers that are located in the front of UDL statements. The use of labels is optional. Labels that are referenced in UDL statements are called tags. The presence of a period (.) in a label implies a UDL statement level. Statements with labels without a period are

Table A-1. UDL Reserved Word List.

ACTION	DECLARE	FIND	LT	POSITION	TAB
ADD	DELETE	FORMAT	MAX	PROCEDURE	TABSET
ALL	DEP	GE	MODE	PULL	THEN
ALONG	DESCEND	GEO	MV	PUT	TO
AND	DISPLAY	GOTO	NKEY	QUIT	TRAILER
ARRAY	DO	GT	NOR	REC	TRANFILE
AT	E	HEADER	NOT	RECORD	TRANS
ATT	EARRAY	HSP	NUM	REMOTE	TREE
BATCH	ECHO	IF	NUMERIC	REMOVE	TRESET
BY	ELSE	IN	OCC	RETURN	USE
CALL	END	INPUT	OCCURRENCE	RGROUP	V
CAT	EPROC	INSIDE	OFF	ROUTE	VAL
CHANGE	EQ	INTER	ON	RTYPE	VALIDATE
CHAR	ERGROUP	INV	OPEN	SAVE	VALUE
CIRCLE	ESCHEMA	KEY	OR	SCHEMA	VERT
CLEAR	ETRAN	LABEL	ORG	SKIP	VIEW
CLOSE	EXECUTE	LE	OUTPUT	SORT	VIS
CONTAINS	EXIST	LINE	OUTSIDE	SOURCE	WORD
CREATE	EXIT	LIST	PAGE	SPACE	WRG
CURRENT	FIELD	LOCAL	POLY	SV	XOR
DATABASE	FILE	LOCSC	POS		

are at statement level-0, statements with one period in their labels are at statement level-1, and so on. Statements without labels assume the level of the first statement with a label that precedes it. The rules governing statement levels with respect to the data manipulation statements are discussed when applicable in paragraph A. 4. 5.

A. 2. 3 Data Constants

This paragraph specifies the syntax and semantics for all legal UDL data constants. UDL recognizes three basic data constants: character-constants, numeric-constants, and geographic-constants. Refer to paragraph A. 3. 2 for discussion of data types.

A. 2. 3. 1 Syntax.

```
<data-constants>        = <character-constant> | <numeric-constant> |
                          <geographic-constant>
<character-constant>    = '<characters>'
<characters>            = <letter> <characters> | <digit> <characters> | <delimiter>
                          <characters> | <letter> | <digit> | <delimiter>
<numeric-constant>      = <fixed-point-constant> | <floating-point-constant>
<fixed-point-constant>  = <sign> <numeric-clause> | <numeric-clause>
<numeric-clause>        = <digit> <numeric-clause> | <numeric-clause> .
                          <numeric-clause> | <numeric-clause> . |
                            <numeric-clause> | <digit>
<sign>                  = + | -
<floating-point-constant> = <fixed-point-constant> E <digit> <digit>
<geographic-constant>   = %<lat-term>/<long-term>
<lat-term>              = <position-term1> <direction1>
<position-term1>        = <degree1> Δ <min-sec> | <degree1>
<degree1>               = <digit> <digit> | <digit>
<min-sec>               = <min> Δ <sec> | <min>
<min>                   = <degree1>
<sec>                   = <degree1>
<direction1>            = N | S
<long-term>             = <position-term2> <direction2>
<position-term2>        = <degree2> Δ <min-sec> | <degree2>
<degree2>               = <digit> <degree1> | <digit>
<direction2>            = E | W
```

A. 2. 3. 2 Semantics. The three UDL data constants are legal in a multitude of UDL statements. For UDL data structures, character-constants are applicable for variable-length-text, fixed-length-text, and alphanumeric data types (refer to paragraph A. 3. 2). Numeric-constants are valid in data structures with numeric data types only, and geographic-constants are valid in data structures with geographic data types only.

A. 2. 4 Expressions

There are two primary expression types recognized in UDL: boolean-expressions and numeric-expressions. Numeric-expressions are divided into two kinds: numeric and restricted numeric. In addition to the normal expressions described in this paragraph, there is another important expression-like construct recognized in UDL called selection-criteria. Selection-criteria is described in paragraph A. 4. Following are the syntax and semantics for UDL expressions.

A. 2. 4. 1 Syntax.

<expression> = <boolean-expression> | <numeric-expression> | <restricted-num-exp>

<boolean-expression> = <boolean-clause> OR <boolean-expression> | <boolean-clause>

<boolean-clause> = <boolean-phrase> AND <boolean-clause> | <boolean-phrase>

<boolean-phrase> = <boolean-factor> <log-op> <boolean-phrase> | <boolean-factor>

<log-op> = XOR | NOR

<boolean-factor> = NOT <boolean-factor> | <boolean-primary>

<boolean-primary> = (<boolean-expression>) | <relational-term>

```
<relational-term>          = <data-spec> <rel-op> <numeric-
                             expression>
<data-spec>                = <field-term> | <variable-term>
<rel-op>                   = LT | GT | LE | GE | EQ
<numeric-expression>       = <char-clause> CAT <numeric-
                             expression> | <char-clause>
<char-clause>              = <numeric-clause> <numeric-op1>
                             <char-clause> | <numeric-clause>
<numeric-op1>              = + | -
<numeric-clause>           = <numeric-phrase> <numeric-op2>
                             <numeric-clause> | <numeric-phrase>
<numeric-op2>              = * | /
<numeric-phrase>           = <numeric-op3> <numeric-phrase> |
                             <numeric-primary>
<numeric-op3>              = + | -
<numeric-primary>          = (<char-clause>) | <numeric-term>
<numeric-term>             = <field-term> | <variable-term> |
                             <data-constant> | <function>
<data-term>                = <field-term> | <array-term> |
                           <repeating-group-term> | <variable-term>
<field-term>               = <field-name> (<subscript-list1>) |
                             <field-name>
<subscript-list1>          = <subscript>, <subscript-list2> |
                             <subscript>
<subscript-list2>          = <subscript>, <subscript> | <subscript>
<subscript>                = <numeric-expression>
<array-term>               = <array-name>
<repeating-group-term>     = <repeating-group-name> (<numeric-
                             expression>) | <repeating-group-name>
```

<variable-term> = @ <variable-name>

<field-name> = <name>

<array-name> = <name>

<repeating-group-name> = <name>

<variable-name> = <name>

<restricted-num-expression> = <restricted-char-clause> CAT <restricted-num-expression> | <restricted-char-clause>

<restricted-char-clause> = <restricted-num-cl> <numeric-op1> <restricted-char-clause> | <restricted-num-cl>

<restricted-num-cl> = <restricted-num-ph> <numeric-op2> <restricted-num-cl> | <restricted-num-ph>

<restricted-num-ph> = <numeric-op3> <restricted-num-ph> | <restricted-num-prim>

<restricted-num-prim> = (<restricted-char-clause>) | <restricted-num-term>

<restricted-num-term> = <variable-term> | <constant> | <restricted-function>

<function> = <function-name> (<function-input-list>) | <function-name>

<function-input-list> = <function-input-param>, <function-input-list> | <function-input-param>

<function-input-param> = <field-term> | <variable-term> | <data-constant>

<restricted-function> = <function-name> (<restricted-input-list>) | <function-name>

<restricted-input-list> = <restricted-input-param>, <restricted-input-list> | <restricted-input-param>

<restricted-input-param> = <variable-term> | <data-constant>

A. 2. 4. 2 Semantics. Boolean-expressions are legal as the condition-clause of an IF statement, and numeric-expressions, both normal and restricted, are valid in many UDL statements. Table A-2 lists all the UDL statements showing legal expression occurrences.

A. 2. 4. 2. 1 Boolean-Expressions. Boolean-expressions may utilize four binary boolean operators and one unary operator in their construction. Following is the evaluation of these five boolean operators.

a. OR – boolean "inclusive-or" condition (union).

A	OR	B
0	0	0
0	1	1
1	1	0
1	1	1

b. AND – boolean "and" condition (intersection).

A	AND	B
0	0	0
0	0	1
1	0	0
1	1	1

c. XOR – boolean "exclusive-or" condition.

A	XOR	B
0	0	0
0	1	1
1	1	0
1	0	1

TABLE A-2. EXPRESSION OCCURRENCES IN UDL STATEMENTS.

Statement	Boolean	Numeric	Restricted Numeric
ADD	–	–	Yes
Assignment	–	Yes	Yes
CALL	–	Yes	Yes
CHANGE	–	–	Yes
CLEAR	–	–	–
CREATE	–	–	Yes
DECLARE	–	–	–
DELETE	–	–	Yes
DISPLAY	–	Yes	Yes
DO	–	Yes	Yes
DO OCCURRENCE	–	–	–
DO RECORD	–	–	–
DO VALUE	–	–	–
FIND	–	–	Yes
FORMAT	–	Yes	Yes
GOTO	–	Yes	Yes
HEADER	–	Yes	Yes
IF	Yes	–	–
OFF	–	–	–
ON	–	–	–
PROCEDURE	–	–	–
PULL	–	–	–
PUT	–	–	–
REMOVE	–	–	–
SORT	–	–	–
TABSET	–	Yes	Yes
TRAILER	–	Yes	Yes

d. NOR – boolean "joint-denial" condition (not-or).

A	NOR	B
0	1	0
0	0	1
1	0	0
1	0	1

e. NOT – boolean "complement" condition (negation).

NOT	A
1	0
0	1

Boolean operator precedence is the following:

a. NOT.
b. XOR, NOR.
c. AND.
d. OR.

The boolean operators XOR and NOR are evaluated "left-to-right." The use of parentheses is valid in boolean-expressions where they force precedence inside their scope. Parentheses may be nested and, if so, evaluation proceeds from the innermost parenthesis level outwards. Boolean operator arguments, termed boolean-primary in the syntax, can either be another boolean-expression if in parentheses, or a relational-term. Relational-terms are formed by placing relational conditions on either field-terms or variable-terms.

Five relational operators are legal in UDL boolean-expressions where they operate as follows:

LT – data-specification less than numeric-expression.

GT – data-specification greater than numeric-expression.

LE – data-specification less than or equal to numeric-expression.

GE – data-specification greater than or equal to numeric-expression.

EQ – data-specification equals numeric-expression.

In boolean-expressions, both user-defined variables and file data structures (fields) are valid data-specification designations for relational conditions.

A. 2. 4. 2. 2 Numeric-Expressions. Five binary operators and two unary operators can be used in constructing numeric-expressions. Following are the definitions of these operators:

a. Binary operators:

+: arithmetic addition.

-: arithmetic subtraction.

*: arithmetic multiplication.

/: arithmetic division.

CAT: character string concatenation.

b. Unary operators:

+: plus sign.

-: minus sign.

Operator precedence is as follows:

a. +, - (unary operators).

b. *, /

c. +, -

d. CAT

The CAT operator has the lowest precedence where it cannot be used as an argument of a numeric operator (see syntax). This implies that conversion from numeric to character is valid but not the reverse. Figure A-1 specifies the legal operator/data type combinations for numeric-expressions.

Parentheses are legal in numeric-expressions where they force operator precedence within their scope.

The subscript-list associated with a field is valid only for fields defined for arrays. Repeating group indexing is not allowed on subordinate fields. Occurrence indexing is legal for repeating groups where the data set level of that repeating group can be controlled (i. e., one subscript). Refer to paragraph A. 3. 1. 2 for more information.

UDL functions are not defined at this time. Their syntax, however, is provided such that their inclusion can be made at a later time.

Restricted-numeric-expressions are identical to numeric-expressions except their numeric-terms cannot contain field-terms. Restricted-numeric-expressions are utilized in UDL update statements (refer to table A-2).

Data Types	Character	Numeric	Geographic
Character	CAT	CAT	CAT
Numeric	CAT	CAT, +, -, *, /	CAT
Geographic	CAT	CAT	CAT

Figure A-1. Legal Numeric Operator/Data Type Combinations.

A.3 DATA DESIGN

This paragraph describes the data structures, data types, and data attributes recognized in UDL. There exist five primary data structures in UDL:

a. Field.
b. Aggregate.
c. Record.
d. File.
e. Data base.

Data types are associated with the lowest level data structure, the field, where three basic types are defined:

a. Character.
b. Numeric.
c. Geographic.

Along with data types, access attributes are also associated with the field, where they control access with respect to interrogation, display, and update.

For each field or aggregate defined, its logical structure is represented with special symbology.

The remainder of this paragraph discusses the foregoing in detail.

A.3.1 Data Structures

Five primary data structures are recognized in UDL: data bases, files, records, aggregates, and fields. The most important structures with respect to user utilization and overall UDL design considerations are fields and aggregates. These two structures will be discussed first.

A. 3. 1. 1 Fields. Two field structures are recognized in UDL:

a. Single-valued field.

b. Multivalued field.

The two fields are discussed in paragraphs A. 3. 1. 1. 1 and A. 3. 1. 1. 2, where their logical structure and rules governing their access with respect to interrogation, display, and update are described.

A. 3. 1. 1. 1 Single-Valued Field. A single-valued field has the logical structure depicted in figure A-2. It has a name with which is associated a single value occurrence. All references to a single-valued field must reference its name. The value associated with a single-valued field must conform to the data-type defined for that field.

A single-valued field can be interrogated by referencing its name against some relational condition where a match is successful depending on its single-valued occurrence. Single-valued fields may have their single value occurrence changed by a CHANGE statement. They cannot, however, be added to or deleted from by the ADD and DELETE statements, which are used exclusively with multioccurring structures.

A. 3. 1. 1. 2 Multivalued Field. A multivalued field has the logical structure depicted in figure A-3. It has a name with which is associated a set of value

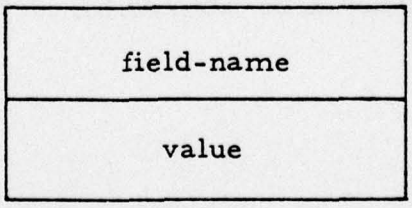


Figure A-2. Logical Structure of a Single-Valued Field.

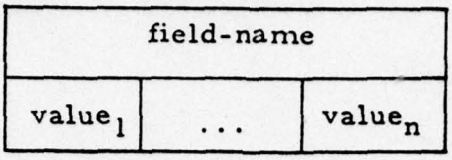


Figure A-3. Logical Structure of a Multivalued Field.

occurrences. All references to a multivalued field must reference its name. Each value occurrence of a multivalued field must conform to the data-type defined for that field. The number of value occurrences that can be contained in a multivalued field is arbitrary, where no occurrences or many occurrences are possible.

A multivalued field can be interrogated by referencing its name against a relational condition where a match is successful depending on a value-by-value search of the value occurrence set contained in the field. Multivalued fields may have a given value occurrence changed to a new value by the CHANGE statement, one or more value occurrences deleted by the DELETE statement, or one or more value occurrences added by the ADD statement. If a multivalued field is displayed where it is not under the control of a DO OCCURRENCE or DO VALUE loop, all values are displayed. If it is under the control of a DO OCCURRENCE loop, that value occurrence currently pointed to by the loop will be displayed. If the multivalued field is under the control of a DO VALUE loop, the unique value which is currently active is displayed. "Under control" means that the multivalued field is the argument source of the DO OCCURRENCE or DO VALUE statement (being inside the loop is not sufficient).

A.3.1.2 Aggregates. Aggregates are a named collection of fields and/or other aggregates. This named collection of data structures can occur an arbitrary number of times within a record. One occurrence of such a collection is called a data set. Since aggregates may contain other aggregates, complex hierarchical structures can be formed with them. A collection of data sets belonging to the same aggregate and having the same ancestral occurrence of a parent aggregate is called an aggregate instance.

In most cases, an aggregate is a structural convenience where the aggregate itself cannot be referenced specifically for the reading or writing of data. There exist exceptions to this, however, where aggregates can be referenced as a totality. Generally these exceptions deal with loop control of aggregate occurrences and with convenience modes in data display. Table A-3 illustrates those cases where aggregate names are legal in UDL statements.

Aggregates do not have specific data types or data attributes associated them. Since an aggregate can have an arbitrary number of fields defined for it, it may have many data types and data attributes. Two aggregate data structures are defined in UDL:

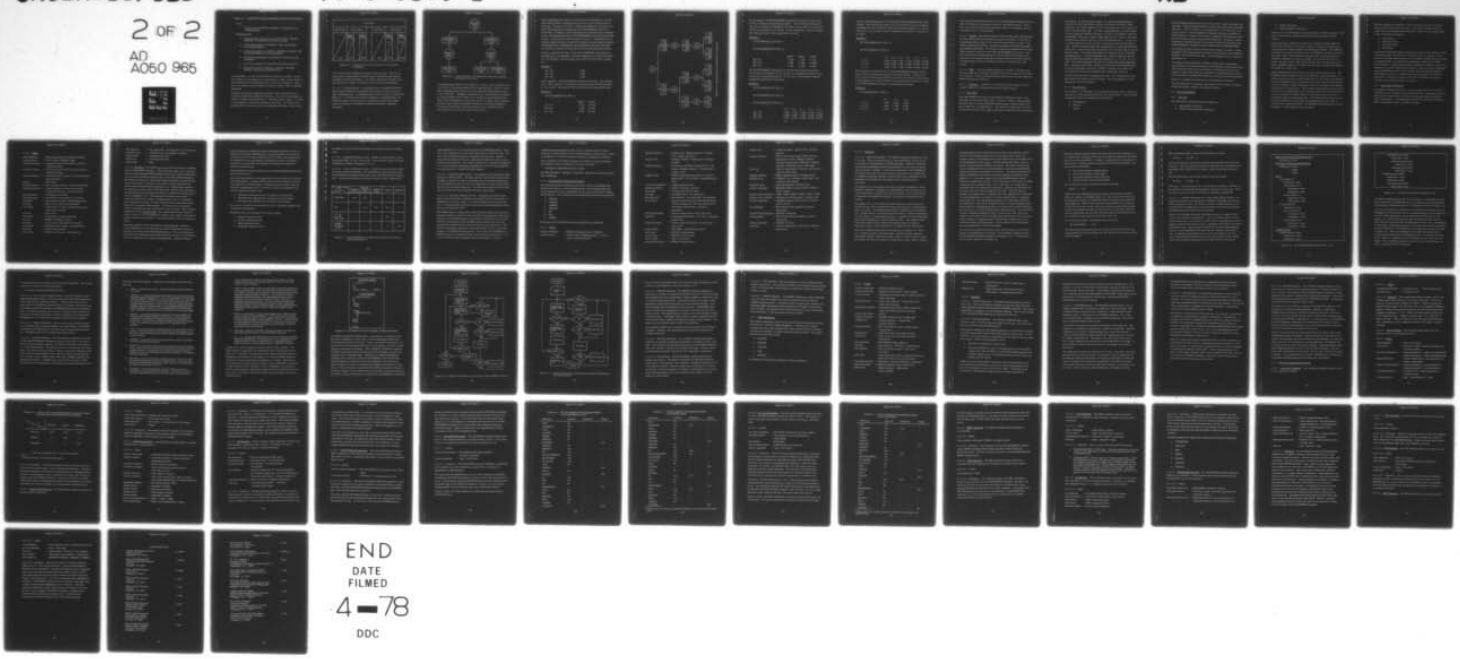
a. Array.
b. Repeating group.

Paragraphs A.3.1.2.1 and A.3.1.2.2 discuss these aggregates in detail.

A.3.1.2.1 Array. An array is a named multivalued data structure which can be composed of any number of fields or arrays. An array has the logical structure depicted in figure A-4. Repeating groups cannot be part of an

AD-A050 965 LOGICON INC SAN DIEGO CALIF F/G 5/2
ADAPT I UNIFORM DATA LANGUAGE (UDL): A FINAL SPECIFICATION.(U)
JAN 78 L E ERICKSON, M E SOLEGLAD N00014-76-C-0899
UNCLASSIFIED 76-C-0899-1 NL

2 OF 2
AD
A050 965



END
DATE
FILMED
4-78
DDC

TABLE A-3. AGGREGATE NAME OCCURRENCES IN UDL STATEMENTS.

Array

1. Display-item in DISPLAY, HEADER, and TRAILER (subscripting illegal).

Repeating Group

1. Scope-qualifier in selection-criteria for FIND, DISPLAY, HEADER, and TRAILER (subscripting illegal).
2. Occurrence-qualifier for CHANGE, ADD, and DELETE (subscripting illegal).
3. Occurrence-control for DISPLAY, HEADER, TRAILER, and DO OCCURRENCE (subscripting optional).
4. Repeating group addition for CREATE and ADD (subscripting illegal).
5. Occurrence-deletion for DELETE (subscripting optional).
6. Max-specification for DISPLAY, HEADER, TRAILER, FORMAT, and DO (subscripting illegal).

array definition. At least one field must be defined in an array. An arbitrary number of occurrences of an array data set is legal, but the number of occurrences must be established as part of the array's definition. Therefore, occurrences cannot be added or deleted by using the ADD or DELETE statements.

An array cannot be interrogated as a totality, where references are restricted to actual fields defined for that array. For each array defined in a record, an index is associated with it. This index has the range 1 – n where n is the number of occurrences defined for that array. All references to fields defined in an array must be accompanied by this index. Since arrays

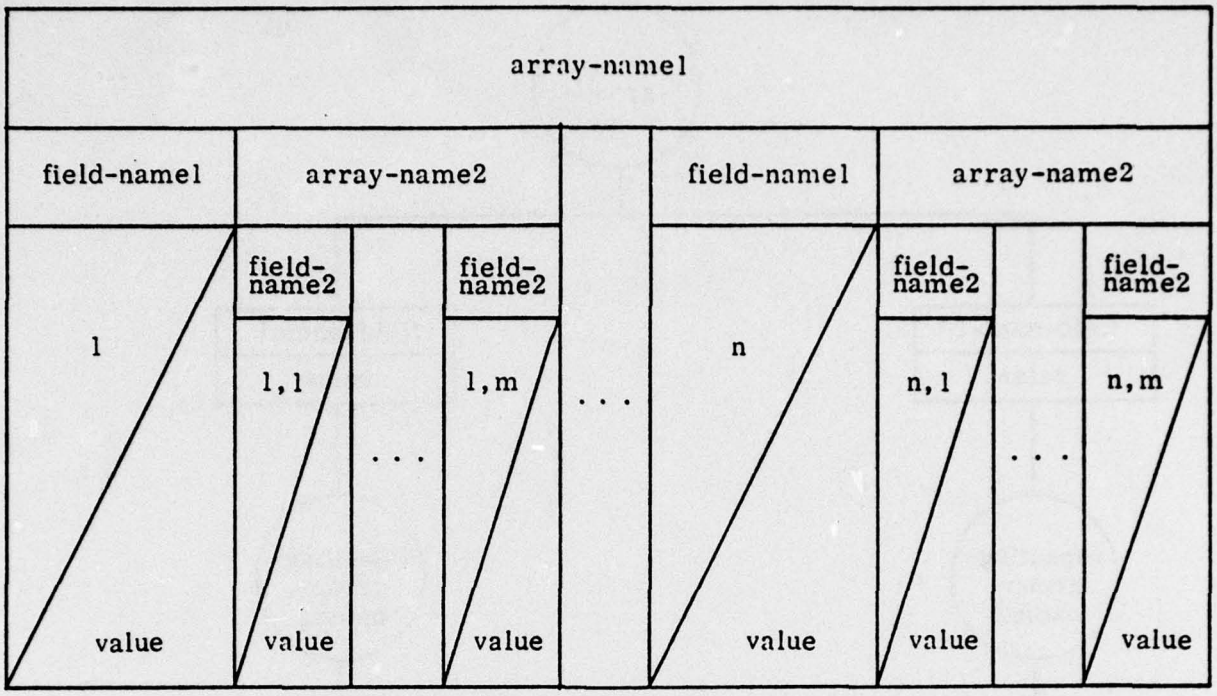


Figure A-4. Logical Structure of an Array (two array levels are shown).

can be nested, other implied subscripts are also required. That is, if a given array is defined under two other arrays, three subscripts must be specified in its field references. Array-name references are legal in display only (refer to table A-3) where an entire array can be displayed.

A.3.1.2.2 Repeating Group. A repeating group is a named multivalued data structure which can be composed of any number of fields, arrays, or other repeating groups. A repeating group's logical structure is depicted in figure A-5. At least one field must be defined in a repeating group. An arbitrary number of occurrences of a repeating group data set is legal, where occurrences can be added or deleted via the ADD or DELETE commands.

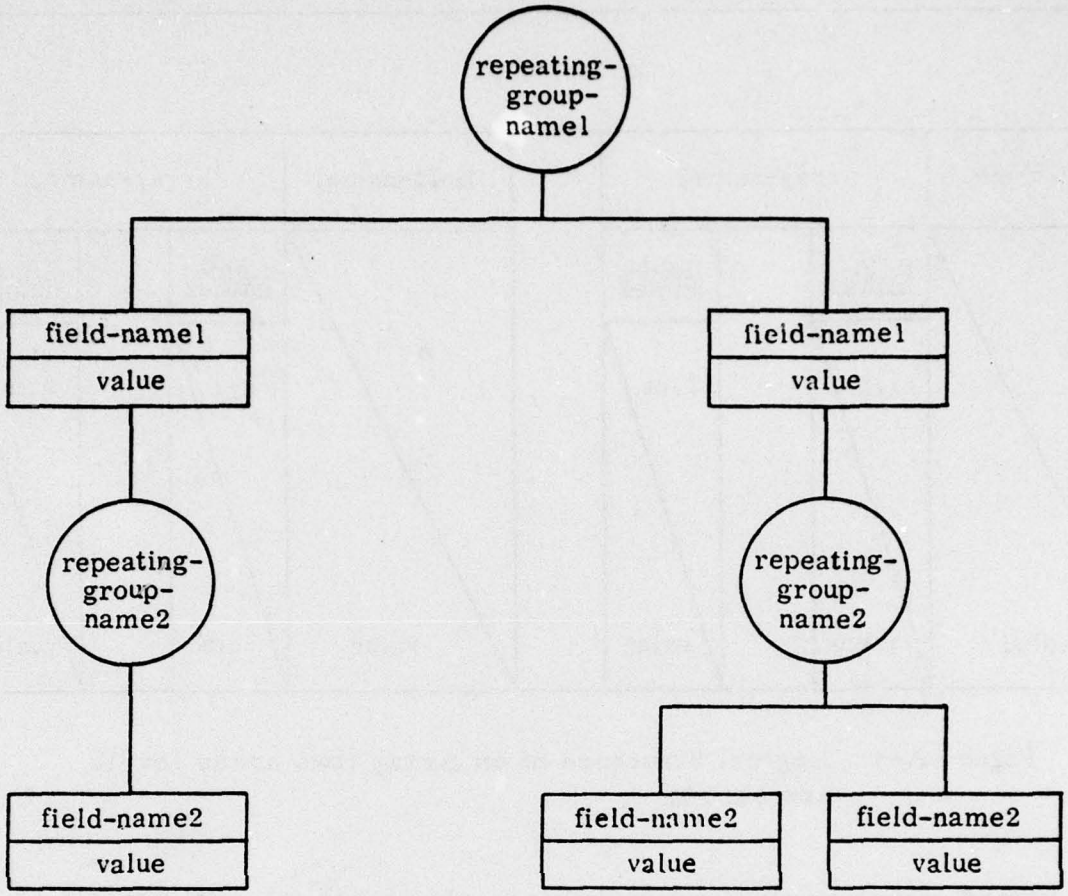


Figure A-5. Logical Structure of a Repeating Group (two repeating group levels are shown).

A repeating group cannot be interrogated as a totality, where references are restricted to actual fields defined for that repeating group. Table A-3 shows where repeating group references (via the repeating-group-name) are legal in UDL statements. Although a repeating group's occurrences can be indexed for some situations (refer to table A-3), an index is not specifically associated with a repeating-group as was the case for an array. Since repeating groups are pure multioccurring data structures, an arbitrary number of occurrences are allowed, where this number may be variable for each record (the size of an array is fixed for all records).

Since repeating groups cannot be referenced for data themselves, only the fields defined within a repeating-group structure can be referenced in expressions. Therefore, in order to transgress through the occurrences of a given repeating group, an appropriate DO OCCURRENCE loop must be established. In addition, if this repeating group is nested in other repeating groups, the appropriate number of DO OCCURRENCE loops (nested) must be present. If less than the required number of loops is established (the appropriate number of loops is equal to the number of nested repeating groups), the ambiguity will be eliminated by the system where only the "left-most" occurrence is accessed (as is the case for referencing multivalued fields).

In order to illustrate the effect of the DO OCCURRENCE statement on the repeating group, six examples of varying complexity are given in the following. All examples utilize the single record data structure shown in figure A-6.

Example 1:

@A = F1:	1→@A
@B = F2:	4→@B
@C = F3:	7→@C

In this example, a DO OCCURRENCE statement is not used. The variables "@A," "@B," and "@C" are assigned a value from three multivalued fields F1, F2, and F3. Note that the "left-most" value is referenced each time.

Example 2:

DO OCCURRENCE RG1 END (α)

⋮

	PASS-1	PASS-2
@A = F1:	1→@A	19→@A
@B = F2:	4--@B	22→@B
@C = F3:	7→@C	25→@C

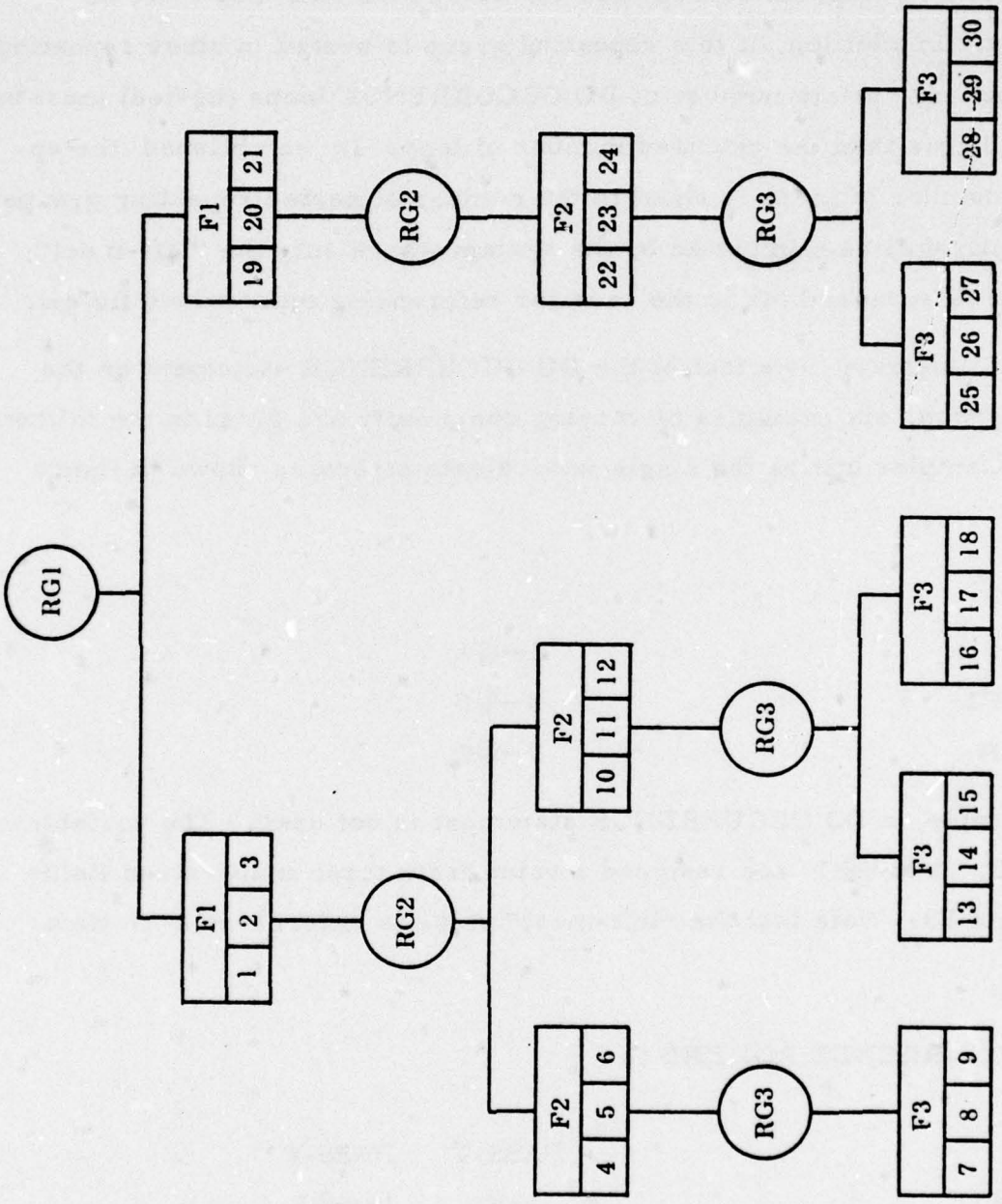


Figure A-6. Repeating Groups and Multioccurring Fields.

In this example, the DO OCCURRENCE statement will loop two times, once for each occurrence of repeating group RG1. For the first pass through the loop, the variables are assigned the "left-most" value of the first occurrence of RG1 for the fields F1, F2, and F3. On the second pass, they are assigned the "left-most-value" of the second RG1 occurrence.

Example 3:

```
DO OCCURRENCE RG1 END (α)
        :
DO OCCURRENCE RG2 END (β)
        :
```

	PASS-1	PASS-2	PASS-3
@A = F1:	1→@A	1 →@A	19→@A
@B = F2:	4→@B	10→@B	22→@B
@C = F3:	7→@C	13→@C	25→@C

Two DO OCCURRENCE statements are utilized in this example where the inside loop goes through all the occurrences of repeating group RG2 for each occurrence of repeating group RG1. In this case, three passes through the assignment statements are exercised.

Example 4:

```
DO OCCURRENCE RG1 END (α)
        :
DO OCCURRENCE RG2 END (β)
        :
DO OCCURRENCE RG3 END (γ)
        :
```

	P-1	P-2	P-3	P-4	P-5
@A = F1:	1→@A	1 →@A	1 →@A	19→@A	19→@A
@B = F2:	4→@B	10→@B	10→@B	22→@B	22→@B
@C = F3:	7→@C	13→@C	16→@C	25→@C	28→@C

All three repeating groups are controlled by DO OCCURRENCE loops in this example. Note now that five passes are exercised where the only ambiguity remaining with value selection is with the multivalued fields F1, F2, and F3. In all cases, the "left-most-value" is selected for each whole occurrence of the field.

Example 5:

```
DO OCCURRENCE RG1 END (α)
        .
        .
        .
DO OCCURRENCE F1 END (β)
        .
        .
        .
```

	P-1	P-2	P-3	P-4	P-5	P-6
A = F1:	1→@A	2→@A	3→@A	19→@A	20→@A	21→@A
B = F2:	4→@B	4→@B	4→@B	22→@B	22→@B	22→@B
C = F3:	7→@C	7→@C	7→@C	25→@C	25→@C	25→@C

This example illustrates the use of a DO OCCURRENCE statement controlling the value occurrence selection of multivalued field F1. Since F1 has three value occurrences for each RG1 occurrence, six passes are exercised. In this example, the "left-most-value" is not selected each time for F1, since the DO OCCURRENCE statement is controlling data selection.

Example 6:

```
DO OCCURRENCE F1 END (α)
        .
        .
        .
```

	P-1	P-2	P-3
A = F1:	1→@A	2→@A	3→@A
B = F2:	4→@B	4→@B	4→@B
C = F3:	7→@C	7→@C	7→@C

This last example illustrates the use of a DO OCCURRENCE statement on the field of F1 without an occurrence control on repeating group RG1. Therefore, since only the "left-most-occurrence" of RG1 is considered, the second instance of F1 is not exercised.

A.3.1.3 Record. Records are probably UDL's second most important data structure where they form the physical unit of selectibility. A record can be optionally named, where if it is named the name refers to a type. Individual records are not named. Records are defined by the fields and aggregates defined under them. All records of a given record-type have the same logical structure where the same fields and aggregates are present. This does not necessarily imply that all records of a given record-type are the same size; true multioccurring data structures such as multivalued fields or repeating groups can have an arbitrary number of occurrences for any record.

A.3.1.4 Files. A file is a named collection of records. A file may contain more than one record-type, where the types must be named if more than one is present. A file is the largest data structure which an interrogation or update statement can reference.

A.3.1.5 Data Base. A data base is a collection of files, where it can be roughly thought of as synonymous with a given target system.

A.3.2 Data Types

Data types designate the nature of the data that a field may contain. UDL data types can be of three basic types: character, numeric, and geographic. Character data are further divided into three groups: variable-length-text, fixed-length-text, and alphanumeric. The further classifications of character data type are based on a usage consideration rather than one of differences

in structure. As will be shown in figure A-7, the CONTAINS operator is valid only for those fields with a character data type of variable or fixed-length-text. For fixed-length-text fields, the EQ operator is also legal (it is not for variable-length-text). If a field is designated with a variable-length or fixed-length-text character data type, the unit of searchability must also be specified. This unit is either a single character or a "word." This unit determines the smallest scan unit of the CONTAINS operator. A "word" unit is usually applicable for files set up for special document retrieval applications. The third character data type, alphanumeric, is valid with the EQ operator only. Numeric data types specify that a field contains data of a purely numeric nature. Two numeric data types are available in UDL: fixed-point and floating-point. For fields designated with a numeric data type, the EQ operator plus all the standard relational operators (i.e., GT, LT, etc.) are valid. The special geographic data type is for fields that contain geographic data. Geographic data always contain the positional dyad latitude/longitude. The special geographic operators, INSIDE, OUTSIDE, and ALONG, are used exclusively with fields designated with a geographic data type.

All UDL fields must have a designated data type. All constants placed into these fields must conform to this data type. Refer to paragraph A. 2. 3 for definition of UDL data constants.

A. 3. 3 Data Attributes

Data attributes, as data types, are associations placed on fields. Data attributes affect a field's operability with the UDL statement set. Three attribute classifications must be designated for a UDL field:

a. Interrogation.

b. Display.

c. Update.

The interrogation attribute can be one of three kinds: keyed, nonkeyed, and dependent. Fields with a data attribute of "keyed" can be referenced in any FIND statement. That is, there are no restrictions to its being used in selection criteria. A field designated as dependent can also be utilized in a FIND statement. However, this must be a dependent FIND statement where its record source is from a previous FIND statement. Therefore, a field designated as dependent cannot be referenced in an initial FIND statement. Fields designated as nonkeyed cannot be referenced in a FIND statement.

The display attribute can be one of two kinds: visible or invisible. Fields designated with a data attribute of visible can be displayed via the DISPLAY, HEADER, and TRAILER statements. Fields designated invisible cannot be displayed at all.

The update attribute determines whether a field is major or nonmajor. Major fields must always be initialized in the CREATE statement and in the ADD statement if they are a major field in a repeating group occurrence. Similarly, major fields which are contained in a repeating group cannot be entirely deleted by the DELETE statement (assuming they are multivalued).

There are no update restrictions for fields designated as nonmajor.

All UDL fields must be assigned a data attribute triad, namely one designating interrogation, display, and update attributes.

A.4 UDL STATEMENTS

A.4.1 Overview

UDL statements are grouped into four categories:

a. Interrogation statements.

b. Display/report generation statements.

c. Update statements.

d. Data manipulation statements.

There is only one interrogation statement in UDL, the FIND statement. FIND selects records from a target system file based on specified selection criteria.

Report generation statements allow the user to obtain a simple display of what he has selected or, at the other extreme, to produce a sophisticated report based on a complex display list and format statement. The DISPLAY statement can be used in conjunction with the FORMAT statement to generate the output. Other UDL statements associated with report generation are TABSET (establishes new user tab settings), ON (activates the output of header and trailer lines), OFF (deactivates the output of header and trailer lines), HEADER (defines the format of a header line), and TRAILER (defines the format of a trailer line).

Update statements allow the UDL user to alter a file in several ways. The REMOVE statement removes entire records from a target system file. To add entire new records to a target system file, the user enters the CREATE statement along with a list of field-name/value pairs which describe the new record to be added. When an existing record is to be changed, the CHANGE statement will accomplish this. The parameters specified are field names of the fields to be changed and the new values for these fields. Two update statements are available to modify records in which there are occurrences of multivalued fields and repeating groups. The ADD statement allows the user to add new field occurrences (under a repeating group) or new value occurrences (in a multivalued field). The DELETE statement does the reverse of ADD; it is used to delete these occurrences.

The data manipulation capability in UDL is integral to the entire UDL language and allows the user to intersperse other UDL statements and commands with the data manipulation statements. There are four general kinds of data manipulation statements:

a. Loop control.
b. Execution control.
c. Data processing.
d. Declaration.

Loop control statements involve the setup and control of loops based on either a specified number, the number of records on a list, the number of occurrences of a specified data structure, or the number of unique values occurring in a specified field. Execution control statements include an IF-THEN-ELSE statement, a GOTO statement (both simple and computed), and a CALL statement (to transfer control to a procedure). There are several data processing statements available for list control, and an assignment statement which assigns a value to a variable based on the evaluation of a numeric-expression. Declaration statements allow the user to name and define a variable, and to name and define a procedure.

A.4.2 Interrogation Statements

Interrogation statements in UDL are used to isolate records from a file by specifying a set of selection criteria against that file. Interrogation does not display a file but only isolates records for subsequent display, updating, and/or manipulation. This paragraph describes the syntax and semantics of UDL's interrogation facility, the FIND statement.

A. 4. 2. 1 Syntax.

```
<find-statement>      = FIND <source-clause> <selection-criteria>
<source-clause>       = IN <file-name> | SOURCE (<tag>)
<selection-criteria>  = <selection-clause> OR <selection-criteria> |
                        <selection-clause>
<selection-clause>    = <selection-phrase> AND <selection-clause> |
                        <selection-phrase>
<selection-phrase>    = <selection-factor> <log-op> <selection-phrase> |
                        <selection-factor>
<log-op>              = XOR | NOR
<selection-factor>    = NOT <selection-factor> | <selection-primary>
<selection-primary>   = <scope-qualifier> (<selection-criteria>) |
                        (<selection-criteria>) | <selection-term>
<scope-qualifier>     = <repeating-group-name>
<selection-term>      = <rel-term1> | <rel-term2> | <rel-term3>
<rel-term1>           = <field-term> <rel-op1> <restricted-num-exp>
<rel-op1>             = EQ | GT | LT | GE | LE | CONTAINS
<rel-term2>           = <field-term> <rel-op2> <restricted-num-exp>,
                        <restricted-num-exp>
<rel-op2>             = WRG | ORG
<rel-term3>           = <field-term> <rel-op3> <geo-exp>
<rel-op3>             = INSIDE | OUTSIDE | ALONG
<geo-exp>             = <circle-exp> | <poly-exp> | <route-exp>
<circle-exp>          = CIRCLE (<radius-term>, <lat-long-term>)
<poly-exp>            = POLY (<lat-long-list>)
<route-exp>           = ROUTE (<band-term>, <lat-long-list>)
```

<lat-long-list> = <lat-long-term>, <lat-long-list> | <lat-long-term>
<lat-long-term> = <variable-term> | <geographic-constant>
<radius-term> = <restricted-num-exp>
<band-term> = <restricted-num-exp>
<source-arg> = <tag>

A. 4. 2. 2 Semantics. The optional source specification must be a statement label of another FIND statement. FIND statements with a source specification are called dependent FIND statements where they isolate records from a set of records previously isolated by one or more other FIND statements. The selection-criteria is a specialized boolean-expression used exclusively for record selection. Its form is very similar in some respects to the boolean-expressions defined in paragraph A. 2. 4. Legal boolean operators for selection-criteria are OR, AND, XOR, NOR, and NOT where they have the same definition and operation precedence as defined for boolean-expressions in paragraph A. 2. 4. The argument of a boolean term (called a selection-primary) can be either a selection-term, which is composed of three relational term types, or other selection-criteria enclosed in parentheses with an optional scope-qualifier. For any given set of selection-criteria, the maximum scope of operation is over a record. This is called a selection-criteria's implicit scope. For complex hierarchical structures within a record, it is also desirable to be able to limit scopes of operation to various subtrees of the structure. The scope-qualifier has been designed to do this.

The scope-qualifier must be the name of a repeating group. Selection-criteria qualified by a scope-qualifier is forced to satisfy that criteria for the same occurrence of the named repeating group. This also implies that the field-terms referenced in the qualified selection-criteria must be defined directly or indirectly for the named repeating group. Indirectly defined

means a field may be defined for another repeating group which, in turn, is defined for the named repeating group. The scope-qualifier will affect the selection process result only if there exist two or more selection-primaries as arguments of the scope-qualifier such that one or more of the logical connectors is an AND (after the selection-criteria has been reduced to disjunctive-normal-form).

Scope-qualifiers can also be nested to force the desired lineage path during the selection process.

Since the scope-qualifier is only applicable for criteria which are connected by an AND operator, complex selection-criteria can be difficult to analyze with respect to its scope-qualifier. Complex selection-criteria under the scope of a repeating group can be reduced to a more understandable form with the application of three rules:

a. Reduce the selection-criteria to disjunctive-normal-form.
b. Distribute the scope-qualifier through the various disjuncts.
c. Drop the scope-qualifier from any single-term disjunct.

The result is a series of scope-qualified terms or unqualified single-terms, connected by OR-operators.

Relational terms can be divided into four basic groups:

a. Standard relational terms.
b. Textscan relational terms.
c. Range relational terms.
d. Geographic relational terms.

See figure A-7 for the valid combinations of relational operators and data types.

A.4.2.2.1 Standard Relational Terms. Standard relational terms contain the operators EQ, LT, GT, LE, and GE. These operators follow the same definition as provided in paragraph A.2.4.2.1.

A.4.2.2.2 Textscan Relational Terms. Textscan relational terms contain the single operator CONTAINS. The CONTAINS operator performs a unit-by-unit scan of the specified field-term for the value depicted by the expression. A field-term must have a data-type of either variable-length-text or

Data Types / Operator	Character			Numeric	Geographic
	Variable-Length-Text	Fixed-Length-Text	Alpha-numeric		
CONTAINS	Yes	Yes	—	—	—
EQ	—	Yes	Yes	Yes	—
LT, GT, LE, GE, WRG, ORG	—	—	—	Yes	—
INSIDE, OUTSIDE, ALONG	—	—	—	—	Yes

Figure A-7. Legal Combinations of Data Types and Selection-Criteria Relational Operators.

fixed-length-text if it is to be referenced by the CONTAINS operator. Similarly, the calculated value of the expression must be a character type. The unit of the CONTAINS operation is specified in the data type. Either a character-by-character search or a data word-by-data word search is performed on the field-term. The "mask" character (?) described in paragraph A. 2. 1 can be utilized in masking out characters in the value represented by the expression (e. g., NAME CONTAINS SCHMI?).

A. 4. 2. 2. 3 Range Relational Terms. Two operators are provided for range operations: WRG and ORG. The WRG operator requires that the field-term be within or equal to the limits established by the two specified expressions. The first expression specifies the lower range limit, and the second expression specifies the upper range limit. The ORG range requires that the field-term be outside the range established by the two expressions.

A. 4. 2. 2. 4 Geographic Relational Terms. Geographic relational terms apply to field-terms (with a geographic data type) and their relationship to a specified set of circles, routes, or n-sided polygons on the surface of the earth. Three geographic operators and three specialized geographic-expressions are recognized in UDL.

The INSIDE operator requires that a field-term's geographic value (latitude/longitude constant) be inside the area specified by the geographic-expression. Similarly, the OUTSIDE operator requires that a field-term's value be outside the area specified, and the ALONG operator requires that the value specified in the field-term be on the perimeter of the specified area. The specialized geographic-expression can specify either a circle, an n-sided polygon, or a multiple directioned route. For a route, the OUTSIDE and INSIDE operators are equivalent where they both mean "off the route." For a circle, both the center and radius must be established. A route is

established by specifying the points it lies on and the width of the route. Polygons are determined by a list of latitude/longitude specifications. Refer to paragraph A. 2. 3 for the syntax and semantics of geographic-constants.

Upon completion of the FIND statement, the number of records satisfying the selection-criteria is displayed to the user as follows:

NUMBER OF RECORDS FOUND = number

All FIND statements, dependent or otherwise, display this record count upon their completion.

A. 4. 3 Display/Report Generation Statements

UDL display statements provide the user with a facility for outputting data extracted from files in a user-oriented format. Data formatting and print control operations can be controlled by the user for the generation of sophisticated reports. UDL recognizes seven display/report generation statements:

a. DISPLAY.
b. FORMAT.
c. HEADER.
d. TRAILER.
e. ON.
f. OFF.
g. TABSET

Following are the syntax and semantics for these seven commands.

A. 4. 3. 1 Syntax.

<display-statement> = DISPLAY <display-clause> | DISPLAY
<display-clause> = <source-spec> <display-phrase1> | <source-spec> | <display-phrase1>

<display-phrase1> = <display-list> <display-phrase2> | <display-list> | <display-phrase2>

<display-list> = <display-element>, <display-list> | <display-element>

<display-element> = <occurrence-qualifier> (<display-list>) | <do-group> | <display-qualifier> < display-item> | <display-item>

<display-item> = TREE (<repeating-group-name>) | <numeric-expression> | <array-term> | <repeating-group-term>

<occurrence-qualifier> = <repeating-group-term>

<display-qualifier> = LOCSC (<selection-criteria>)

<do-group> = (<display-list> DO <do-spec>)

<do-spec> = <variable-term> = <do-range-list>

<do-range-list> = <do-element>, <do-range-list> | <do-element>

<do-element> = <numeric-expression> TO <max-specification> | <numeric-expression> TO <max-specification> BY <numeric-expression> | <numeric-expression>

<max-specification> = <numeric-expression> | MAX (<rgn-list>)

<rgn-list> = <repeating-group-name>, <rgn-list> | <repeating-group-name>

<display-phrase2> = <format-spec> <dest-spec> | <format-spec> | <dest-spec>

<format-spec> = USE (<tag>) | <format-declaration>

<dest-spec> = REMOTE (<remote-arg>)

<source-spec> = SOURCE (<source-arg>)

<source-arg> = <tag> | <list-name>

<format-declaration> = FORMAT (<format-list>)

<format-list> = <format-element>, <format-list> | <format-element>

<format-element> = PAGE | LINE (<line-arg>) | SKIP (<numeric-expression>) | TAB | TRESET | SPACE (<numeric-expression>) | RETURN | <display-activate> | <iteration-spec> | USE (<tag>)

<line-arg> = <numeric-expression>, <numeric-expression> | <numeric-expression>

<display-activate> = VERT (<display-term>) | <display-term>

<display-term> = CURRENT | AT (<numeric-expression>) | TO (<numeric-expression>)

<iteration-spec> = <max-specification> (<format-list>)

<header-statement> = HEADER <header-trailer-number> <header-trailer-clause>

<header-trailer-number> = <digit> <digit> <digit> | <digit> <digit> | <digit>

<header-trailer-clause> = <display-list> <format-spec> | <display-list>

<trailer-statement> = TRAILER <header-trailer-number> <header-trailer-clause>

<on-statement> = ON <header-trailer-qualifier> <numeric-expression>

<header-trailer-qualifier> = HEADER | TRAILER

<off-statement> = OFF <header-trailer-qualifier> <numeric-expression>

<tabset-statement> = TABSET <tab-list>

<tab-list> = <numeric-expression>, <tab-list> | <numeric-expression>

A.4.3.2 Semantics.

A.4.3.2.1 DISPLAY Statement. The DISPLAY statement operates on a list of records. These records may be on a user-specified list or they may be on an implicit list generated by the FIND statement. DISPLAY statements may operate independently where they specify the record source via the optional parameter, SOURCE, or they may be embedded in a DO RECORD or DO VALUE loop where the record source is specified by these statements. If a DISPLAY statement specifies a record source and is embedded in a DO RECORD or DO VALUE loop, the explicit source specification has precedence.

If a display-list is not specified, the entire record(s) is displayed. If the DISPLAY statement is embedded in a DO RECORD or DO VALUE loop, the entire record that is current in the loop is displayed. If the record source is specified in the DISPLAY statement, all records are displayed in their entirety.

The display-list is a list of data structures below the record level which provide the user with a selective and ordered mechanism for displaying portions of a record. The fundamental element of a display list is the display-item. Display-items can be one of three basic kinds: a numeric-expression, an array-term, or a repeating-group-term. In addition, an entire repeating group may be displayed by using the TREE qualifier. The display of field-terms and variable-terms is allowed through the display of numeric-expressions where a single field-term or variable-term are valid forms.

Other more sophisticated constructs are allowed in a display list. These are: a display-qualifier, DO-loop facility, and an occurrence-qualifier. The display-qualifier, denoted by LOCSC, allows the user to further qualify the selected records with respect to a single display-item. That is, before a

specified display-item is output for a given record, the selection-criteria denoted as the display-qualifier must be satisfied by that record. If a display-qualifier is not specified for a given display-item, that display-item will be displayed for each record. The DO-loop facility allows a user to embed a DO-loop inside a DISPLAY statement. The scope of the DO-loop may be a display-list which is specified prior to the DO-loop specification inside the "do-group" parentheses (see syntax). The DO-loop specification follows essentially the same form as the DO statement described in paragraph A. 4. 5. A user may specify a list of do-elements where each do-element is satisfied left-to-right. A do-element may consist only of a single component hence causing the loop to occur only once, or a range may be specified where the variable-term specified is indexed by one starting from the initial value up to and including the final value. If the BY option is utilized, the variable-term is indexed by the value specified in its argument. The max-specification can either be a numeric-expression or a repeating-group-name list. The repeating-group-name list is used when a DO-loop encompasses all occurrences currently present in a set of repeating groups. If more than one repeating-group-name is specified, they must be defined at the same level; i. e., have the same parent repeating group. The max-specification is evaluated dynamically where it takes on the value equal to the number of occurrences of the repeating group with the most occurrences for a given instance (a repeating group may have many instances if it is subordinate to another repeating group). As the instances are transgressed, the max-specification is altered accordingly.

The occurrence-qualifier provides the user with the facility to control the display of repeating group occurrences. Its operation is analogous to the DO OCCURRENCE statement defined in paragraph A. 4. 5. The argument of an occurrence-qualifier is a display-list.

Since the argument scope for both a DO-block and occurrence-qualifier can be a display-list, all display items types are legal (including other DO-blocks and/or occurrence-qualifiers). In order to specify the precise operation of the occurrence-qualifier and how it interacts with the DO-loop facility, four basic cases are described:

a. Occurrence-qualifier (unsubscripted).
b. Occurrence-qualifier (subscripted).
c. Occurrence-qualifier inside a DO-loop.
d. DO-loop inside an occurrence-qualifier.

The occurrence-qualifier (unsubscripted) takes the format:

rgn (X_1, ..., X_n)

In this case, the repeating-group-name (rgn) is an occurrence-qualifier which, for a given set of display-items, $X_1 - X_n$, forces value selection to be performed in a group corresponding to the occurrences of the specified repeating group. That is, value selection is performed for X_1 through X_n for the first occurrence of rgn, then value selection is again performed for all X_i for the second occurrence of rgn, and so on. If the value selection were not qualified by rgn, the value selection would be performed on X_1 for all occurrences of rgn, then all occurrences of X_2, etc.

The occurrence-qualifier (subscripted) takes the format:

rgn (subscript)(X_1, ..., X_n)

This case behaves as the previous case except value selection for $X_1 - X_n$ is performed for only one occurrence of rgn, namely the occurrence pointed to by "subscript."

The occurrence-qualifier inside a DO-loop takes the format:

(rgn (X_1, ..., X_n) DO ...)

This case illustrates a nested loop condition where the DO-loop is the "outside loop." For each index value taken on by the DO-loop, value selection is performed on $X_1 - X_n$ for all occurrences of rgn as described in the first case.

The DO-loop inside an occurrence-qualifier takes the format:

rgn ((X_1, ..., X_n DO ...))

This case, as in the previous case, is a nested loop condition. However, in this situation, the repeating group qualifier forms the "outside loop." Hence, for each occurrence of rgn, the DO-loop is completely transgressed.

A.4.3.2.1.1 General Comments on the UDL Default Display Format. If a format-specification is not specified in a DISPLAY statement, the UDL default format will be used. This format provides a simple means for a user to display fields and aggregates of records without undo concern about data formatting and general print control procedures.

Figure A-8 illustrates the actual UDL default format as it applies to the various kinds of fields and aggregates. Due to the inherent nature of hierarchical structures, records which contain levels of repeating groups will be displayed with the tree structure intact, regardless of the order of the specified display-items. Data structures defined for the same repeating group, however, will be displayed in the order they are specified in the display-list. A repeating group display-item will cause the display of all occurrences (assuming local selection criteria are not specified on this display-item) of the specified repeating group at its data set level only; that is,

Single-Valued Field/Concatenated Field

```
field-name = value
```

Multivalued Field/Concatenated Field

```
field-name = value1
           = value2
             :
           = valuen
```

Array

```
array-name1 (1)
    field-name = value
    array-name2 (1, 1)
        field-name = value
        field-name = value
    array-name2 (1, 2)
        field-name = value
        field-name = value
array-name1 (2)
    field-name = value
    array-name2 (2, 1)
        field-name = value
        field-name = value
    array-name2 (2, 2)
        field-name = value
        field-name = value
```

Repeating Group

```
Repeating group-name1
    field-name = value
    field-name = value
```

Figure A-8. UDL Default Display Format (Sheet 1 of 2).

```
            Repeating group-name2
                field-name = value
                repeating group-name3
                    field-name = value
                repeating group-name3
                    field-name = value
        Repeating group-name1
            field-name = value
            field-name = value
            repeating group-name2
                field-name = value
```

Figure A-8. UDL Default Display Format (Sheet 2 of 2).

subordinate repeating groups will not be displayed. A repeating group display-item qualified by TREE behaves as the situation just described except all subordinate repeating groups are also displayed (as shown in figure A-8).

If a list of field-terms is specified as a set of display-items where they are embedded in a hierarchical structure of repeating groups, the minimum tree structure which encompasses all the specified field-terms will be displayed. Repeating group names will also be displayed as node markers, where they will be inserted into the proper places of the tree. Combinations of field-terms, repeating-group-terms, and array-terms may be specified in the same display-list. The display for each display-item behaves as shown in figure A-8. Each record displayed will contain the minimum tree structure which encompasses the displayed-items. As brought out earlier in this section, if a display-list is not specified, the entire record is displayed. Thus if repeating groups are present in the record definition, the entire tree structure will be displayed.

A record header will be displayed at the top of each print page. This record header will contain the following information:

(file-name) RECORD NUMBER IS number

A. 4. 3. 2. 1. 2 User-Directed Display Format. If the user desires a more sophisticated display output, he may circumvent the UDL default display format by specifying a format-specification. Format-specifications can be made either in the DISPLAY statement itself or the format-specification can be remotely specified by pointing to a FORMAT statement via its statement label. Format-specifications can be used in conjunction with a display-list, or they can be used with the entire record if a display-list is not specified. paragraph A. 4. 3. 2. 2 elaborates on display-list/format-list interaction.

A. 4. 3. 2. 1. 3 Display Destination. The user may specify where the output is to be displayed by utilizing the optional dest-specification. If REMOTE is specified, the output will go to one of the system's high-speed line printers. The default case is the requesting input/output console.

A. 4. 3. 2. 2 FORMAT Statement. The UDL FORMAT statement used in conjunction with the DISPLAY statement provides the user with a very sophisticated report generation capability. A FORMAT statement is composed of a list of format-elements. Format-elements can be divided into three basic types: those which control printlines, spacing, and general tabulation; those which cause the actual display of data; and those which control the format-list. Of the first type, PAGE, LINE, SKIP, TAB, TRESET, SPACE, and RETURN are present. CURRENT, AT, and TO (with or without a VERT qualifier) are of the second type, and iteration-specification and USE are the

third type of format-elements. Following is a description of those format-elements.

a. PAGE – causes a top-of-form. All line/character position pointers are reset.

b. LINE (X, Y) – causes the line position pointer to be set to line X and character position pointer to Y. If Y is not specified, the character position pointer is unaltered. For the condition where X is less than the current line position, a top-of-form is performed and the line position pointer is set to X. If Y is less than the current character position, the line position pointer is incremented and the character position pointer is set to Y. As is implied by the foregoing conditions, UDL does not allow the user to "backup" a line or print page.

c. SKIP (X) – causes the line position pointer to be incremented by X. The character position pointer is not altered. If the current line position plus X exceeds the maximum line pointer allowed for a print page, a top-of-form is performed and the line position pointer is set to (current-line position pointer) +X- (maximum-line-position).

d. TAB – causes the character position pointer to be set to the next tab position. If the tab pointer is pointing to the last tab position, it is then set to the first tab position (cyclic) and the line position pointer is incremented.

e. TRESET – causes the character position pointer and the tab pointer to be set to the first tab position.

f. SPACE (X) – causes the character position pointer to be incremented by X. If the value exceeds the maximum character position, the line position pointer is incremented by one and the character position pointer is set to (current-character position pointer) +X- (maximum-character-position).

g. RETURN – causes the character position pointer to be set to 1 and the line position pointer to be incremented by one. If the line position pointer exceeds the maximum, a top-of-form is performed and the line position pointer is set to 1.

h. CURRENT – causes display of the current display-item, left-justified at the current character position. If in the course of displaying the specified display-item the character position pointer

equals the maximum position, the line position pointer is incremented and the character-position pointer is set to 1 where the display is then continued.

i. AT(X) – causes display of the current display-item left-justified at character position X. If X is less than the current character position pointer, the line position pointer is incremented by one and the character position pointer is set to X. If X is greater than the maximum character position, the line position pointer is incremented by one and the character position pointer is set to X - (maximum character position). If in the course of displaying the specified display-item the character position pointer equals the maximum position, the line position pointer is incremented and the character position pointer is set to 1 where the display is then continued.

j. TO(X) – causes display of the current display-item, right-justified to the current character position X. If X is less than the current character position pointer, the line position pointer is incremented by one and the character position pointer is set to X. If X is greater than the maximum character position, the line position pointer is incremented by one and the character position pointer is set to X - (maximum-character-position).

k. USE (tag) – causes the FORMAT statement pointed to by tag to be utilized at this point in the current FORMAT statement.

l. X (. . .) – causes repeated usage of the format-elements contained within the parentheses. The format-list will be repeated X-times. X may be either a numeric-expression or a max-specification. Refer to discussion of max-specification in DISPLAY statement.

Display-terms may be optionally qualified with the VERT qualifier. The VERT qualifier is applicable with multivalued fields or named multivalued fields only. Normally, multivalued fields and named multivalued fields are displayed in a horizontal list (see figure A-9). When the VERT qualifier is specified, the values of the field are displayed in a column. Each value is left-justified to the same character position if AT or CURRENT is specified, or right-justified if TO is specified. If AT or CURRENT is specified, the final character position is that position following the value with the most characters in it. If TO (X) is specified, the final character position is the position following X.

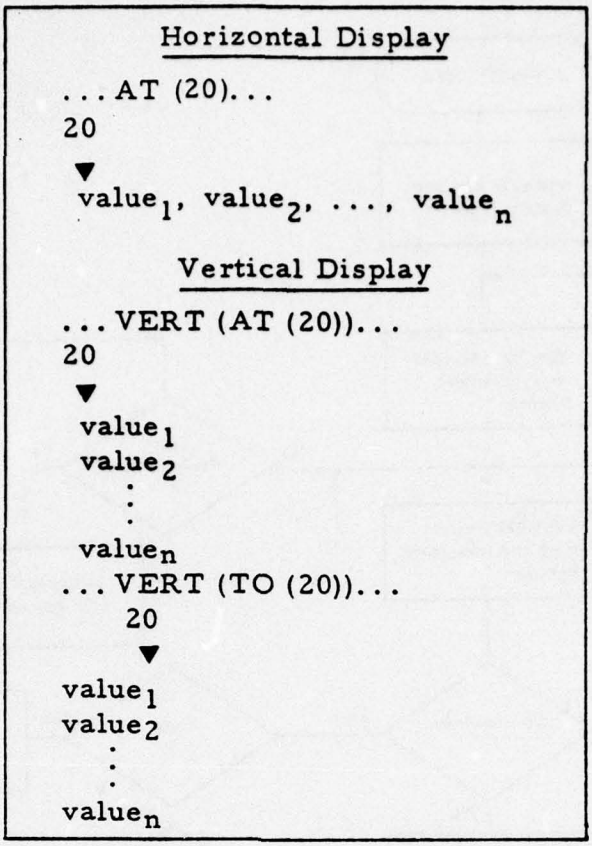


Figure A-9. Horizontal and Vertical Display of Multivalued Fields.

For DISPLAY statements which specify a record source, the display-list is processed completely for each record. The accompanying format-list is processed cyclically for all records. That is, the format-list pointer is not reset to the first format-element for each record (unless, of course, the list happens to terminate at this point). When a DISPLAY statement is entered, its display-list pointer and the associated format-list pointer are set to the beginning of their respective lists. Therefore, if a DISPLAY/FORMAT statement pair is under the control of a DO RECORD or DO VALUE loop, both list pointers are reset for each record displayed. The flow charts shown in figures A-10 and A-11 illustrate this display-list/format-list interaction. If a record source is specified in the DISPLAY statement, the process shown in figure A-10 is entered once and terminates when all records are processed. If the record source is controlled by a DO RECORD or DO VALUE

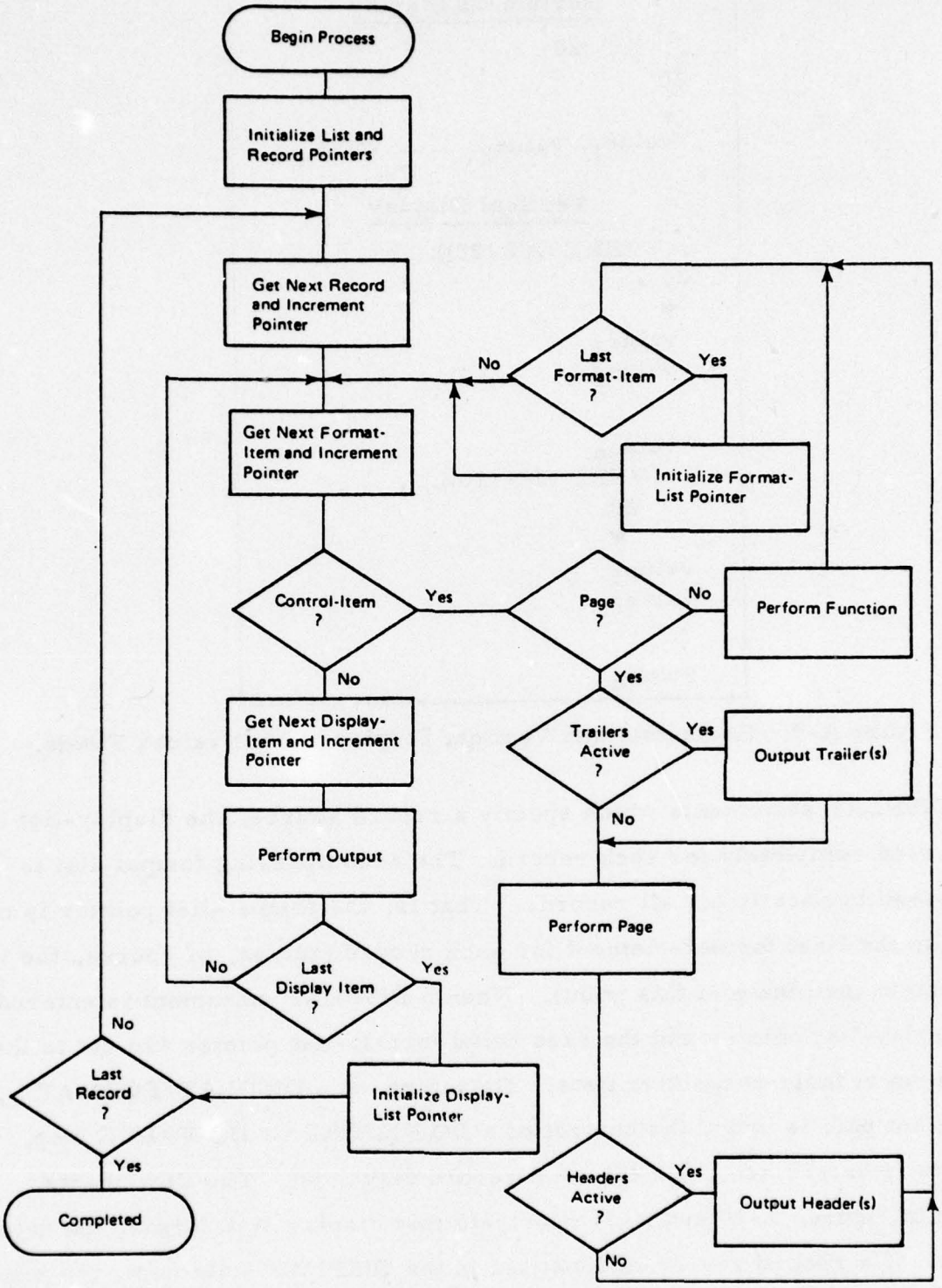


Figure A-10. Display-List/Format-List Interaction Under DISPLAY Control.

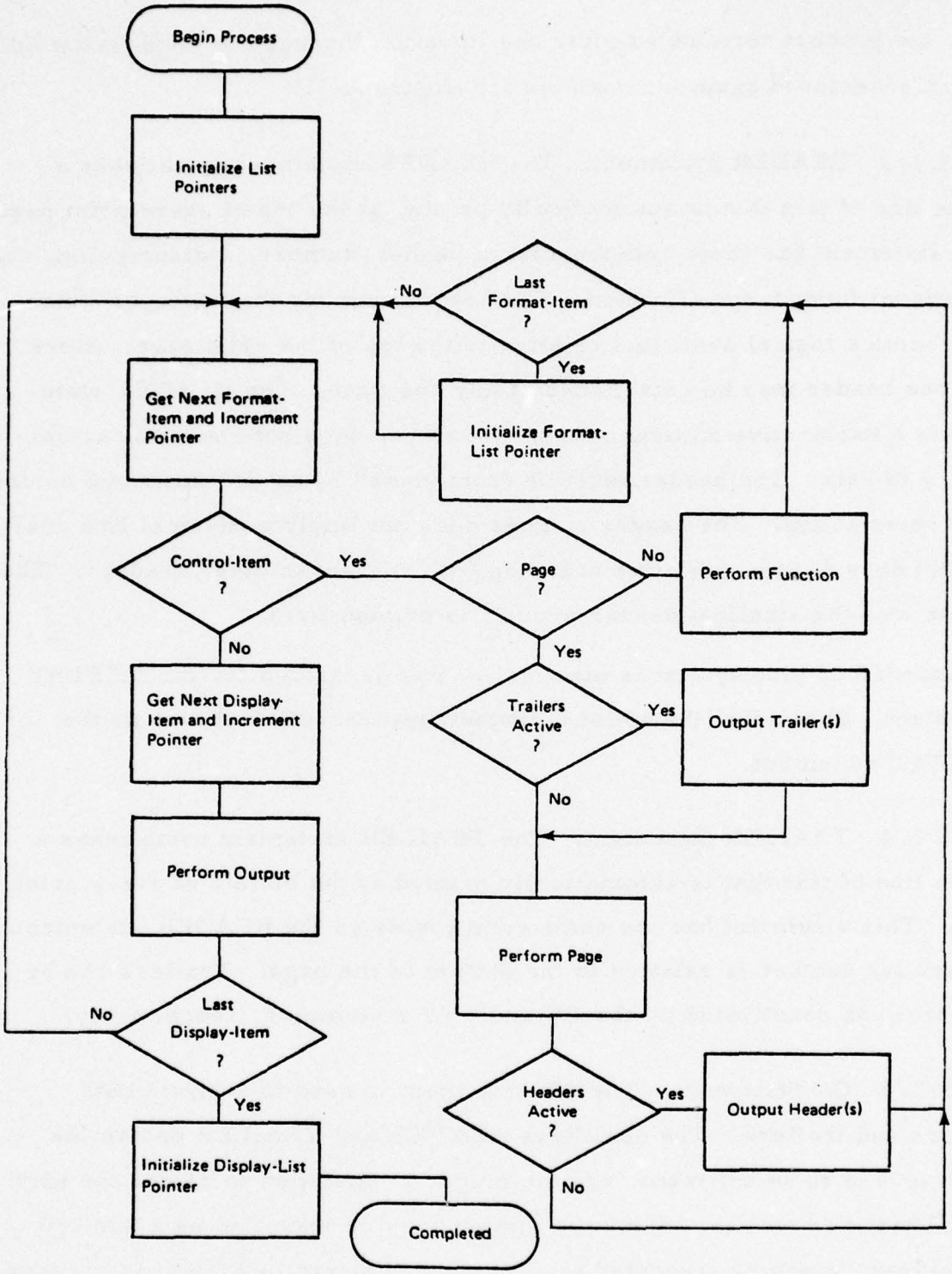


Figure A-11. Display-List/Format-List Interaction Under DO RECORD/ DO VALUE Control.

loop, the process terminates after one iteration through the display-list and is then re-entered again for each record (figure A-11).

A.4.3.2.3 HEADER Statement. The HEADER statement establishes a single line of text that is automatically printed at the top of every print page. This statement has three components: a header-number, a display-list, and an optional format-specification. The header-number is an integer which represents a logical print line relative to the top of the print page. More than one header may be established at any one time. The HEADER statement is a declarative statement and, therefore, does not cause the actual display of data. The header must be "turned-on" by an ON statement before it becomes active. The header number does not imply a physical line position but does denote a relative ordering with respect to other headers. The header with the smallest header number is printed first.

The mandatory display-list is identical to that described for the DISPLAY statement. Similarly, the optional format-specification behaves as the FORMAT statement.

A.4.3.2.4 TRAILER Statement. The TRAILER statement establishes a single line of text that is automatically printed at the bottom of every print page. This statement has the same components as the HEADER statement. The trailer number is relative to the bottom of the page. Trailers can be activated and deactivated by the ON and OFF statements, respectively.

A.4.3.2.5 ON Statement. The ON statement is used to activate both headers and trailers. The qualifiers HEADER and TRAILER determine which type is to be activated, and an integer is specified to denote the particular header or trailer. Since the integer may be specified as a numeric-expression, it will be truncated to an integer if a fractional part is present. If the resulting integer is not a valid header/trailer number, an error diagnostic will be output.

A.4.3.2.6 OFF Statement. The OFF statement is used to deactivate both headers and trailers. The qualifiers HEADER and TRAILER determine which type is to be deactivated, and an integer is specified to denote the particular header or trailer.

A.4.3.2.7 TABSET Statement. The TABSET statement is used to establish a set of column tab settings to be used in conjunction with the TAB and TRESET format elements. Tab settings must be integers and, therefore, the evaluated number-expression will be truncated to an integer if a fractional part is present. Any number of tab settings may be set, where they must be in the range from 1 to the maximum character position.

A.4.4 Update Statements

UDL update statements provide the user with an interactive data base maintenance facility. With UDL update statements, a user may add or delete records from a file, alter existing field values, add new value occurrences to multioccurring fields, or delete existing value occurrences. Five update statements are provided in UDL:

a. REMOVE.

b. CREATE.

c. CHANGE.

d. ADD.

e. DELETE.

Following are the syntax and semantics for these statements.

A. 4. 4. 1 Syntax.

```
<remove-statement>     = REMOVE SOURCE (<tag>)
<create-statement>     = CREATE IN <file-name> <create-clause>
<create-clause>        = RTYPE (<record-type-name>) <field-value-list> |
                         <field-value-list>
<field-value-list>     = <field-value-element> , <field-value-list> |
                         <field-value-element>
<field-value-element>  = <repeating-group-name> (<field-value-list>) |
                         <field-term> EQ <restricted-num-exp>
<record-type-name>     = <name>
<change-statement>     = CHANGE SOURCE (<tag>) <change-list>
<change-list>          = <change-element> , <change-list> |
                         <change-element>
<change-element>       = OCC (<selection-criteria>) <change-part> |
                         <change-part>
<change-part>          = <field-spec> TO <restricted-num-exp>
<field-spec>           = <field-name> EQ <restricted-num-exp> |
                         <field-term>
<add-statement>        = ADD SOURCE (<tag>) <add-list>
<add-list>             = <add-element> , <add-list> | <add-element>
<add-element>          = OCC (<selection-criteria>) <add-item> |
                         <add-item>
<add-item>             = <repeating-group-name> (<restricted-add-list>) |
                         <field-term> EQ <restricted-num-exp>
<restricted-add-list>  = <add-item>, <restricted-add-list> | <add-item>
<delete-statement>     = DELETE SOURCE (<tag>) <delete-list>
<delete-list>          = <delete-element> , <delete-list> |
                         <delete-element>
```

<delete-element> = OCC (<selection-criteria>) <delete-part> |
<delete-part>

<delete-part> = <field-name> EQ <restricted-num-exp> |
<field-term> | <repeating-group-name>

A. 4. 4. 2 Semantics.

A. 4. 4. 2. 1 REMOVE Statement. The REMOVE statement deletes records from a file. The records must be isolated by a previous FIND statement whose statement label is referenced by the REMOVE statement. This source specification is mandatory for the REMOVE statement. All records isolated by the referenced FIND statement are deleted from the file.

A. 4. 4. 2. 2 CREATE Statement. The CREATE statement adds a new record to a file. If the file contains multiple record types, the record-type must be specified via the RTYPE qualifier.

The primary component of the CREATE statement is a field-value-list which is used to assign initial values to the fields defined for that record. In most situations, it is not necessary to assign values to all fields in order to create a new record. There are two exceptions to this:

a. The field is designated as a major field.

b. The fields contained in a parent repeating group must be assigned a value(s) in order for fields in a subordinate repeating group to be assigned values; that is, a repeating group occurrence cannot exist disjoint from the record.

A field-value-list has two types of elements: those which establish value occurrences for a hierarchical structure (a tree of repeating groups) and those which establish a value occurrence for a field. A repeating group occurrence is established by stating its name and then specifying the

appropriate field value occurrences defined for that repeating group. This process, of course, can be nested if subordinate repeating groups are to be initialized with value occurrences. Fields must be initialized with data that are compatible with the data type defined for those fields. Field values may be expressed as restricted-numeric-expressions, hence great flexibility is afforded the user in record creation.

A. 4. 4. 2. 3 CHANGE Statement. The CHANGE statement allows a user to change existing value occurrences in fields for a set of records. The records must have been previously isolated by a FIND statement whose statement label is referenced by the CHANGE statement. The record source specification is mandatory for the CHANGE statement. All records have the specified changes made to them.

The primary component of the CHANGE statement is the change-list. The change-list is composed of change-parts, qualified or unqualified. Qualified change-parts are valid for repeating groups only, where the selection-criteria inside the OCC parameter isolates the repeating group occurrences the change is applicable for. Depending on the selectivity of the criteria, one or many repeating group occurrences may be affected by the change. If no occurrences satisfy the selection-criteria, the change is not made for that record.

The change-part specifies the field and its new value. The value, which may be expressed as a restricted-numeric-expression, must be compatible with the data-type defined for that field. For multioccurring fields, the desired value occurrence to be changed may be specified by denoting its old value. If more than one value occurrence equals this old value, it is changed to the new value. If no occurrences equal the old value, no changes are made.

If a multivalued field is referenced in a change-part where no old value occurrence is specified, all old occurrences are deleted and the field is set to a new value where it becomes its only value occurrence.

A. 4. 4. 2. 4 ADD Statement. The ADD statement allows a user to add new value occurrences to repeating groups or multivalued fields for a set of records. The records must have been previously isolated by a FIND statement whose statement label is referenced by the ADD statement. The record source specification is mandatory for the ADD statement. All specified records will have the additions made to them.

The primary component of the ADD statement is the add-list. The add-list is composed of add-elements, qualified or unqualified. Qualified add-elements are valid for repeating groups only, where the selection-criteria inside the OCC parameter isolates the repeating group occurrence the addition is applicable for. As was the case with the CHANGE statement, depending on the selectivity of the criteria, one or many repeating group occurrences may be affected. If no occurrences satisfy the selection criteria, the addition is not made for that record.

Either new value occurrences to multivalued fields or new occurrences to a hierarchical structure (a tree of repeating groups) may be added. If a new value occurrence is to be added to a multivalued field, it must be compatible with the data-type defined for that field. If a new repeating group occurrence is to be added, all major fields must be initialized with data. As was the case with the CREATE statement, this process may be nested where many levels of repeating groups can be added.

A. 4. 4. 2. 5 DELETE Statement. The DELETE statement allows a user to delete value occurrences from repeating groups or multivalued fields for a set of records. The records must have been previously isolated by a FIND statement whose label is referenced by the DELETE statement. The record source specification is mandatory for the DELETE statement. All specified records will have the deletions made to them.

The delete-list is the primary component of the DELETE statement. The delete-list is composed of delete-elements, qualified or unqualified. Qualified delete-elements are valid for repeating groups only, where the selection-criteria inside the OCC parameter isolates those repeating group occurrences the deletion is applicable for. As was the case for the CHANGE and ADD statements, depending on the selectivity of the criteria, one or many occurrences may be affected. If no occurrences satisfy the selection-criteria, the deletion is not made for that record.

Either a value occurrence of a multivalued field or an entire occurrence of a repeating group may be deleted. If all the occurrences of a multivalued field are to be deleted, only the field-name must be specified. If a selected value occurrence is to be deleted, it must be specified. The deletion will occur only if the old value occurrence is present. Multivalued fields occurring in repeating groups may have value occurrences deleted. However, if this field is declared as a major field, all of its value occurrences cannot be deleted. If a repeating group occurrence is deleted, all occurrences of subordinate repeating groups of that instance are also deleted.

A. 4. 5 UDL Data Manipulation Statements

A. 4. 5. 1 Assignment Statement. The assignment statement allows a value to be assigned to a variable.

A. 4. 5. 1. 1 Syntax.

```
<assignment-statement>   = <variable-term> = <numeric-expression>
<variable-term>          = @ <variable-name>
```

A. 4. 5. 1. 2 Semantics. The assignment statement assigns a value to the variable based on the evaluation of a numeric-expression. UDL recognizes two variable modes: implicit and explicit. Implicit variables are defined by their usage as the "left-hand-side" of an assignment statement. Explicit variables are previously defined by the user with the DECLARE statement (paragraph A. 4. 5. 4). Implicit variables assume the data type of the "right-hand-side." Explicit variables take on the scaling that is defined for them. Table A-4 shows the valid "left-hand-side/right-hand-side" combinations for the assignment statement with respect to explicit variables.

A. 4. 5. 2 CALL Statement. The CALL statement enables the user to transfer control to a predefined procedure.

A. 4. 5. 1 Syntax.

```
<call-statement>           = CALL <call-clause>
<call-clause>              = <procedure-name> <procedure-parameters>|
                             <procedure-name>
<procedure-parameters      = <input-parameters>, <output-exit-parameters>|
                             <input-parameters> | <output-exit-parameters>
<input-parameters>         = <numeric-expression>, <input-parameters> |
                             <numeric-expression>
<output-exit-parameters>   = <output-parameters>, <exit-parameters>|
                             <output-parameters> | <exit-parameters>
<output-parameters>        = <variable-term>, <output-parameters> |
                             <variables-term>
<exit-parameters>          = <tag>, <exit-parameters> | <tag>
```

TABLE A-4. LEGAL "LEFT-HAND-SIDE/RIGHT-HAND-SIDE" COMBINATIONS FOR THE ASSIGNMENT STATEMENT.

L \ R	Character	Numeric	Geographic
Character	Yes	Yes[1]	Yes[1]
Numeric	—	Yes	—
Geographic	—	—	Yes

(where the "left-hand-side" is an explicit variable)

[1] "Right-hand-side" is converted to character.

A. 4. 5. 2. 2 Semantics. The CALL statement will cause control to be transferred to the named procedure, with accompanying parameters. The procedure will be executed, and control will be returned to the calling procedure or batch. The parameters must be specified in the same order as they are defined in the PROCEDURE statement. Control is returned via an EXIT statement to either the statement following the CALL statement or to another statement whose label is specified as an exit parameter (abnormal exit).

A. 4. 5. 3 CLEAR LIST Statement. The CLEAR LIST statement removes all records from the named list.

A. 4. 5. 3. 1 Syntax.

```
<clear-list-statement> = CLEAR LIST <clear-list-clause>
<clear-list-clause>    = <list-name-list> | ALL
<list-name-list>       = <list-name> , <list-name-list> | <list-name>
<list-name>            = <name>
```

A. 4. 5. 3. 2 Semantics. The named list does not disappear from existence when this statement is executed; it only becomes clear of record images. If ALL is specified, all lists for this user are cleared.

A. 4. 5. 4 DECLARE Statement. The DECLARE statement defines a variable and describes its characteristics.

A. 4. 5. 4. 1 Syntax.

```
<declare-statement>         = DECLARE <variable-name> <declare-clause>
<declare-clause>            = <character-phrase> | <numeric-phrase> |
                              <geographic-phrase>
<character-phrase>          = CHAR <character-element>
<character-element>         = <size-specification> <preset-value1> |
                              <size-specification>
<numeric-phrase>            = NUM <numeric-element>
<numeric-element>           = <size-scale-specification> <preset-value2> |
                              <size-scale-specification> | E <preset-value3>|E
<geographic-phrase>         = GEO <preset-value4> | GEO
<preset-value1>             = <character-constant>
<preset-value2>             = <fixed-point-constant>
<preset-value3>             = <floating-point-constant>
<preset-value4>             = <geographic-constant>
<size-scale-specification>  = <digit> / <digit> | <digit>
<size-specification>        = <digit> <size-specification> | <digit>
```

A. 4. 5. 4. 2 Semantics. The name of the variable is specified along with its type (character, numeric, or geographic). For variables designated as character, a size-specification is mandatory, where it specifies the maximum number of characters the variable may contain. Fixed-point variables must also have a size specified where it denotes the maximum number of decimal digits the variable may contain. An optional scaling specification may be placed on fixed-point variables, where it denotes the number of fractional decimal digits for that variable. All variables may be preset with an appropriate data-constant.

A. 4. 5. 5 DO Statement. The DO statement allows execution of a UDL statement sequence a specified number of times. The number of times is specified via a series of values a variable will assume.

A. 4. 5. 5. 1 Syntax.

```
<do-statement>          = DO <do-specification> END (<tag>)
<do-specification>      = <variable-term> = <do-element-list>
<do-element-list>       = <do-element> , <do-element-list> |
                          <do-element>
<do-element>            = <numeric-expression> TO <max-specification>
                          BY <numeric-expression> | <numeric-expression>
                          TO <max-specification> | <numeric-expression>
< max-specification>    = MAX (<rgn-list>) | <numeric-expression>
<rgn-list>              = <repeating-group-name> , <rgn-list> |
                          <repeating-group-name>
```

A. 4. 5. 5. 2 Semantics. The DO statement establishes a loop or a set of loops based on the values a specified variable will assume. A single loop is defined by its initial value, its final value, and the increment used as the loop is transgressed. If an increment is not specified, an increment of one

is assumed as a default value. If a final value is not specified, the loop is processed exactly once, where the variable is set to the specified initial value. The max-specification can be either a numeric-expression or a list of repeating-groups. If MAX is specified, the final value assumes the maximum occurrence of the repeating-group list. Refer to paragraph A. 4. 3. 2 for more information on the MAX qualifier.

The scope of the DO loop is a set of UDL statements starting with the UDL statement immediately following the DO statement and ending with the statement pointed to by tag. All statements within the DO loop assume a statement level one greater than the DO statement, and, therefore, if a user assigns statement labels within a DO loop, they must reflect this condition.

A. 4. 5. 6 DO OCCURRENCE Statement. The DO OCCURRENCE statement allows execution of a UDL statement sequence a specified number of times. The number of times is specified by the number of occurrences in a specified data structure.

A. 4. 5. 6. 1 Syntax.

<do-occurrence-statement> = DO OCCURRENCE <occurrence-clause> END (<tag>)

<occurrence-clause> = <field-term> | <repeating-group-term>

A. 4. 5. 6. 2 Semantics. The DO OCCURRENCE statement establishes a loop based on the number of occurrences contained in a specified field or repeating group. The DO OCCURRENCE statement must be inside a DO RECORD loop, which establishes the record source.

The scope of the DO OCCURRENCE loop is a set of UDL statements immediately following the DO OCCURRENCE statement and ending with the statement pointed to by tag. All statements within the DO OCCURRENCE loop

assume a statement level one greater than the DO OCCURRENCE statement, and, therefore, if a user assigns statement labels within the DO loop, they must reflect this condition. In the UDL data structure description (paragraph A.3.1.2.2), six examples are presented which show the use of the DO OCCURRENCE statement and its interaction with repeating groups and multivalued fields. Table A-5 shows which UDL statements are affected when placed inside a DO OCCURRENCE loop.

A. 4. 5. 7 DO RECORD Statement. The DO RECORD statement allows execution of a UDL statement sequence a specified number of times. The number of times is specified by the number of records contained on a list.

A. 4. 5. 7. 1 Syntax.

<do-record-statement> = DO RECORD END (<tag>) SOURCE
(<source-argument>)

<source-argument> = <tag> | <list-name>

A. 4. 5. 7. 2 Semantics. The DO RECORD statement establishes a loop based on the number of records contained on a named list or an implicit list generated by a FIND statement.

The scope of the DO RECORD loop is a set of UDL statements immediately following the DO RECORD statement and ending with the statement pointed to by tag. All statements within the DO RECORD loop assume a statement level one greater than the DO RECORD statement; therefore, if a user assigns statement labels within the DO loop, they must reflect this condition.

Table A-6 shows which UDL statements are affected when placed inside a DO RECORD loop.

TABLE A-5. UDL STATEMENT OCCURRENCES INSIDE A DO OCCURRENCE LOOP.

Statement	Affected	Mandatory	Illegal
ADD	No	-	-
Assignment	Yes	-	-
CALL	No	-	-
CHANGE	No	-	-
CLEAR	No	-	-
CREATE	No	-	-
DECLARE	-	-	Yes
DELETE	No	-	-
DISPLAY	Yes	-	-
DO	Yes	-	-
DO OCCURRENCE	Yes	-	-
DO RECORD	Yes	-	-
DO VALUE	Yes	-	-
FIND	No	-	-
FORMAT	-	-	Yes
GOTO	No	-	-
HEADER	-	-	Yes
IF	Yes	-	-
OFF	No	-	-
ON	No	-	-
PROCEDURE	-	-	Yes
PULL	No	-	-
PUT	No	-	-
REMOVE	No	-	-
SORT	Yes	-	-
TABSET	No	-	-
TRAILER	-	-	Yes

TABLE A-6. UDL STATEMENT OCCURRENCES INSIDE A DO RECORD LOOP.

Statement	Affected	Mandatory	Illegal
ADD	No	-	-
Assignment	Yes	Yes[1]	-
CALL	No	-	-
CHANGE	No	-	-
CLEAR	No	-	-
CREATE	No	-	-
DECLARE	-	-	Yes
DELETE	No	-	-
DISPLAY	Yes	-	-
DO	Yes	Yes[1]	-
DO OCCURRENCE	Yes	Yes	-
DO RECORD	Yes	-	-
DO VALUE	Yes	-	-
FIND	No	-	-
FORMAT	-	-	Yes
GOTO	No	-	-
HEADER	-	-	Yes
IF	Yes	Yes[1]	-
OFF	No	-	-
ON	No	-	-
PROCEDURE	-	-	Yes
PULL	Yes	Yes	-
PUT	Yes	-	-
REMOVE	No	-	-
SORT	Yes	-	-
TABSET	No	-	-
TRAILER	-	-	Yes

[1] If data-names are referenced and the statement is not inside a DO VALUE loop.

A. 4. 5. 8 DO VALUE Statement. The DO VALUE statement allows execution of a UDL statement sequence a specified number of times. The number of times is specified by the number of unique values occurring in a specified field.

A. 4. 5. 8. 1 Syntax.

<do-value-statement> = DO VALUE <field-term> <do-value-clause>
<do-value-clause> = <end-clause> <source-specification> |
<end-clause>
<end-clause> = END (<tag>)
<source-specification> = SOURCE (<source-argument>)
<source-argument> = <tag> | <list-name>

A. 4. 5. 8. 2 Semantics. The DO VALUE statement establishes a loop based on the number of unique value occurrences contained in a multivalued field. The DO VALUE statement may optionally be inside a DO RECORD loop, where the record source is established and controlled. If the DO VALUE statement is not inside a DO RECORD loop, the record source must be specified in the DO VALUE statement. Any reference to the multivalued field specified in the DO VALUE statement by a statement inside the DO VALUE loop will reference the value occurrence that is currently active.

The scope of the DO VALUE loop is a set of UDL statements immediately following the DO VALUE statement and ending with the statement pointed to by tag. All statements within the DO VALUE loop assume a statement level one greater than the DO VALUE statement, and, therefore, if a user assigns statement labels within the DO loop, they must reflect this condition.

Table A-7 shows which UDL statements are affected when placed inside a DO VALUE loop. The value-set looped on is determined either from all values

TABLE A-7. UDL STATEMENT OCCURRENCES INSIDE A DO VALUE LOOP.

Statement	Affected	Mandatory	Illegal
ADD	No	-	-
Assignment	Yes	Yes[1]	-
CALL	No	-	-
CHANGE	No	-	-
CLEAR	No	-	-
CREATE	No	-	-
DECLARE	-	-	Yes
DELETE	No	-	-
DISPLAY	Yes	-	-
DO	Yes	Yes[1]	-
DO OCCURRENCE	Yes	-	-
DO RECORD	Yes	-	-
DO VALUE	Yes	-	-
FIND	No	-	-
FORMAT	-	-	Yes
GOTO	No	-	-
HEADER	-	-	Yes
IF	Yes	Yes[1]	-
OFF	No	-	-
ON	No	-	-
PROCEDURE	-	-	Yes
PULL	No	-	-
PUT	No	-	-
REMOVE	No	-	-
SORT	Yes	-	-
TABSET	No	-	-
TRAILER	-	-	Yes

[1] If data-names are referenced and the statement is not inside a DO RECORD loop.

the field contains in a single record if the DO VALUE statement is under the control of a DO RECORD loop, or all values the field contains in all records. For the latter case, the DO VALUE statement specifies its own record source.

A. 4. 5. 9 EPROC Statement. The EPROC statement ends the definition of a procedure.

A. 4. 5. 9. 1 Syntax.

<end-procedure-statement> = EPROC <procedure-name>

A. 4. 5. 9. 2 Semantics. All statements between the PROCEDURE statement and the EPROC statement with matching procedure-names become the defined procedure. All UDL procedures must have a matching PROCEDURE/ EPROC statement pair.

A. 4. 5. 10 EXIT Statement. The EXIT statement allows control to be returned from a procedure back to the batch or procedure which called it.

A. 4. 5. 10. 1 Syntax.

<exit-statement> = EXIT

A. 4. 5. 10. 2 Semantics. If the statement label on the EXIT statement is named as an exit parameter in the PROCEDURE statement, the EXIT is an abnormal exit from the procedure and control is returned to the corresponding statement label specified in the CALL statement. If the label on the EXIT statement is not named as an exit parameter in the PROCEDURE statement, it is a normal exit and control is returned to the statement following the CALL to the procedure.

A. 4. 5. 11 GOTO Statement. The GOTO statement changes execution control from one sequence of UDL statements to another sequence of UDL statements.

A. 4. 5. 11. 1 Syntax.

<go-to-statement> = GOTO <go-to-clause>
<go-to-clause> = <tag> | <computed-go-to-phrase>
<computed-go-to-phrase> = (<tag-list>) <numeric-expression>
<tag-list> = <tag> , <tag-list> | <tag>

A. 4. 5. 11. 2 Semantics. There are two forms of the GOTO statement:

a. Unconditional GOTO: GOTO <tag>. When this statement is executed, control is unconditionally transferred to the statement whose label is specified by tag.

b. Computed GOTO: GOTO (<tag-list>) <numeric-expression>. When this statement is executed, the numeric-expression is evaluated, the result is truncated to an integer value, and this result is used as an index into the tag-list. If the truncated result of the numeric-expression is less than one or greater than the number of tags in the list, control is transferred to the next statement. Otherwise, control is transferred to the statement whose label in the list corresponds to the truncated value of the expression.

A. 4. 5. 12 IF Statement. The IF statement allows execution of a UDL statement sequence if a specified boolean expression is satisfied, or a different UDL statement sequence if the boolean expression is not satisfied.

A. 4. 5. 12. 1 Syntax.

<if-statement> = IF <boolean-expression> <then-else-block>
<then-else-block> = <then-block> <else-block> | <then-block>
<then-block> = THEN <statement-block>
<else-block> = ELSE <statement-block>
<statement-block> = "block of UDL statements"

A. 4. 5. 12. 2 Semantics. If the boolean-expression is satisfied, the then-block is executed. If the boolean-expression is not satisfied, either the else-block is executed (if specified) or the first statement following the IF statement with the same level as the IF statement is executed. Both the then-block and the else-block must exist at one statement level below that of the IF statement, hence all user-specified labels in this block must reflect this condition.

All UDL statements are legal in the statement-block except the following:

a. PROCEDURE.

b. DECLARE.

c. EXIT.

d. EPROC.

e. HEADER.

f. TRAILER.

g. FORMAT.

A. 4. 5. 13 PROCEDURE Statement. The PROCEDURE statement begins the definition of a procedure. It also specifies the name of the procedure and the parameters used in the procedure.

A. 4. 5. 13. 1 Syntax.

```
<procedure-statement>    = PROCEDURE <procedure-clause>
<procedure-clause>       = <procedure-name> <procedure-parameters> |
                           <procedure-name>
<procedure-parameters>   = <input-parameters> <out-exit-parameters> |
                           <input-parameters> | <out-exit-parameters>
```

```
<input-parameters>        = INPUT (<input-parameter-list>)
<out-exit-parameters>     = <output-parameters> <exit-parameters> |
                            <output-parameters> | <exit-parameters>
<output-parameters>       = OUTPUT (<output-parameter-list>)
<exit-parameters>         = EXIT (<tag-list>)
<input-parameter-list>    = <numeric-expression>, <input-parameter-
                            list> | <numeric-expression>
<output-parameter-list>   = <variable-term>, <output-parameter-list> |
                            <variable-term>
<tag-list>                = <tag>, <tag-list> | <tag>
```

A. 4. 5. 13. 2 Semantics. All the statements between the PROCEDURE statement and the EPROC statement with matching procedure names become the defined procedure. Procedures cannot be defined inside other procedures. There are three types of parameters that may be specified with the PROCEDURE statement: input, output, and exit (abnormal). Input parameters are indicated via the INPUT key word and supply input values to the procedure. Input values may be simple data-constants, fields-names, or complex numeric-expressions. Output parameters are indicated via the OUTPUT key word and will carry output values from the execution of the procedure. Output parameters are limited to variables. Exit parameters are indicated via the EXIT key word and specify abnormal exits from the procedure. An abnormal exit is one in which control does not return to the statement following the CALL to the procedure but to some other statement whose label is specified in the CALL statement. Although the parameters specified in the CALL statement are not preceded by the key words INPUT, OUTPUT, and EXIT, these parameters must be in the same order as the parameters defined for the procedure (in the PROCEDURE statement).

A. 4. 5. 14 PULL Statement. The PULL statement removes a record from a list.

A. 4. 5. 14. 1 Syntax.

<pull-statement> = PULL

A. 4. 5. 14. 2 Semantics. The PULL statement must be inside a DO RECORD loop. The record to be removed from the list is a single record which is the current record in a DO RECORD loop. If this is the only record on the list, the list becomes empty but is still defined for the user.

A. 4. 5. 15 PUT Statement. The PUT statement places a record(s) on a list.

A. 4. 5. 15. 1 Syntax.

<put-statement> = PUT <put-clause>
<put-clause> = <list-name> SOURCE (<source-argument>) |
<list-name>
<source-argument> = <tag> | <list-name>
<list-name> = <name>

A. 4. 5. 15. 2 Semantics. The record(s) to be placed on the list named by list-name is either a single record which is the current record in a DO RECORD loop, or is one or more records pointed to by source-argument. If the list does not already exist at the time this statement is executed, it is created at this time.

A. 4. 5. 16 SORT Statement. The SORT statement sorts a list of records.

A. 4. 5. 16. 1 Syntax.

<sort-statement>	=	SORT SOURCE (<source-argument>) <key-list>
<source-argument>	=	<tag> | <list-name>
<key-list>	=	<key-element>, <key-list> | <key-element>
<key-element>	=	<field-term> <key-qualifier> | <field-term>
<key-qualifier>	=	DESCEND NUMERIC | DESCEND | NUMERIC

A. 4. 5. 16. 2 Semantics. The list of records to be sorted is either an implicit list (i. e. , the records isolated by a previous FIND statement, named as source-argument) or an explicit list named as source-argument. The order is specified by the key-list which implies a major-to-minor order where the next key is used only to further sort equal records with respect to the previous key. The order is ascending unless DESCEND is specified. The NUMERIC qualifier is used for character type fields when a numeric rather than an alphanumeric sort is desired. If the field specified in field-term is multi-valued, the first occurrence is used for the sort. Upon completion of the SORT statement, an implicit list is constructed which contains the sorted records. This list can be referenced by the SORT statement label, which must be present.

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, VA 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2 copies
Office of Naval Research Code 102IP Arlington, VA 22217	6 copies
Office of Naval Research Code 200 Arlington, VA 22217	1 copy
Office of Naval Research Code 455 Arlington, VA 22217	1 copy
Office of Naval Research Code 458 Arlington, VA 22217	1 copy
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1 copy
Office Of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1 copy

New York Area Office — 1 copy
715 Broadway – 5th Floor
New York, NY 10003

Naval Research Laboratory — 6 copies
Technical Information Division, Code 2627
Washington, D. C. 20375

Dr. A. L. Slafkosky — 1 copy
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D. C. 20380

Naval Electronics Laboratory Center — 1 copy
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner — 1 copy
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, MD 20084

Captain Grace M. Hopper — 1 copy
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350

Mr. Kin B. Thompson — 1 copy
Technical Director
Information Systems Division (OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20305

Advanced Research Projects Agency — 1 copy
Information Processing Techniques
1400 Wilson Boulevard
Arlington, VA 22209